Санкт-Петербургский государственный университет

*Сорина Полина Олеговна*

**Выпускная квалификационная работа**

# *Молекулярно-термодинамическое моделирование образования везикул в водно-солевых растворах ионных поверхностно-активных веществ*

Уровень образования: *бакалавриат*
Направление: *04.03.01 «Химия»*
Основная образовательная программа: *СВ.5014.2017 «Химия»*
Профиль: «Физико-химический»

Научный руководитель:
профессор кафедры физической химии, д. х. н., проф. Викторов Алексей Исмаилович

Рецензент:
доцент кафедры коллоидной химии, к. х. н. Ванин Александр Александрович

Санкт-Петербург
2021

**Содержание**

## Введение

Везикулы представляют собой замкнутые бислойные агрегированные структуры, окруженные извне раствором и содержащие раствор в своей внутренней части. Растворы везикул – типичные представители широкого класса систем, называемого мягкой материей. Особенность таких систем - способность откликаться на небольшие внешние воздействия радикальной перестройкой их мезомасштабной структуры, в результате чего может происходить существенное изменение макроскопических свойств системы. Чувствительность к изменениям в среде обусловливает применение таких систем в разработке так называемых «умных материалов» [1,2]. Везикулы образуются в растворах многих классических и природных поверхностно-активных веществ (ПАВ) и блоксополимеров [3,4]. Благодаря своим уникальным свойствам, системы с везикулами широко используются в бытовой химии, косметике и медицине [5]. Помимо везикул, в растворах ПАВ образуются агрегаты и других форм: сферические и червеобразные мицеллы, глобулы, диски, тороидальные агрегаты, плоские бислои и пространственные сетки. При этом определяющее влияние на тип и размер агрегатов оказывает строение и концентрация молекул ПАВ, а также такие факторы, как температура и состав растворителя, в частности, присутствие электролита, и его концентрация, pH раствора.

Большинство везикул имеет вид сферического бислоя. В водных растворах везикул ПАВ гидрофильные вещества могут проникнуть во внутреннюю область везикулы, тогда как гидрофобные молекулы способны проникать в мембрану везикулы, размещаясь среди хвостов ПАВ. Таким образом, везикулы могут выступать в качестве капсул и контейнеров и используются для транспортировки веществ [6].

Исследование и синтез липосом [7], которые представляют собой везикулы, состоящие из амфифильных молекул фосфолипидов, имеет важное значение в биологии. Моделирование везикул, образованных ПАВ более простого строения, может служить как прототип для выявления механизмов процессов, происходящих в амфифильных мембранах, образованных сложными молекулами.

Строгое статистико-механическое рассмотрение агрегации в растворах ПАВ, весьма затруднительно, ввиду сложности этих систем. Поэтому наиболее продуктивным подходом, позволяющим связать молекулярные характеристики ПАВ с агрегативным поведением раствора, являются молекулярно-термодинамические модели [8–10], сочетающие элементы статистико-механического описания с эмпирическими соотношениями.

В настоящее время большое внимание уделяется катанионным системам. Их использование позволяет относительно просто управлять мезомасштабной структурой и физико-химическими свойствами раствора путем варьирования соотношения разных ПАВ в растворе, солевого фона и кислотности.

Важнейшей характеристикой везикул и мембран, в особенности катанионных, является трансмембранный потенциал - разность электрических потенциалов на внешней и внутренней поверхностях мембраны. Трансмембранный потенциал является движущей силой переноса подвижных ионов через мембрану, поэтому возможность регулировать величину и знак трансмембранного вызывает большой интерес. В связи с этим особенно важной задачей является поиск аналитического выражения для расчета трансмембранного потенциала.

Таким образом, целью данной работы было аналитическое описание электростатической составляющей свободной энергии агрегации и трансмембранного потенциала, исследование поведения трансмембранного потенциала в зависимости от солевого фона раствора и геометрических параметров везикулы, а также разработка алгоритмов и написание программного обеспечения для моделирования образования везикул в катанионных растворах ПАВ в присутствии соли с учетом возможности образования агрегатов других форм.

## 2. Обзор литературы

### 2.1. Везикулы в растворах ПАВ и их молекулярно-термодинамическое описание

Агрегация молекул ПАВ в везикулы наблюдается как самопроизвольно, так и в результате стимулирующего воздействия, например, под влиянием ультразвука. Образование везикул наблюдается в растворах двухвостых ПАВ, блок-сополимеров, фосфолипидов [3,11]. Форма двухвостых молекул ПАВ является благоприятной для образования искривленного бислоя, что отражается значением фактора упаковки [12], который будет рассматриваться нами в этом разделе. Самопроизвольная агрегация везикул наблюдается в растворах катанионных однохвостых ПАВ, где наличие голов противоположных знаков позволяет создать ионную пару. Такая ионная пара имеет строение, аналогичное двухвостому ПАВ, что позволяет им образовывать стабильные везикулы.

Основными экспериментальными методами для обнаружения агрегатов различной морфологии являются криогенная трансмиссионная электронная микроскопия [13], малоугловое рассеяние нейтронов (SANS) [14], а также малоугловое рассеяние рентгеновских лучей (SAXS) [15]. Примерами систем, в которых наблюдаются везикулы, являются водные растворы смеси алкилсульфата натрия (SOS, SDS) с бромидом алкилтриметиламмония (CTAB, DTAB) [16,17], смеси додецилбензолсульфоната натрия (SDBS) с различными алкилтриметиламмониевыми и пиридиниевыми катионными ПАВ (DTAB, CTAT, CPyCl, DTAC) [18] и смеси поверхностно-активных катионных ионных жидкостей таких, как хлорид 1-додецил-3-метилимидазолия ($C_{12}$mimCl), хлорид 1-гексадецил-3-метилимидазолия ($C_{16}$mimCl), с анионными ПАВ (SDS, SDBS) [19]. Катанионные везикулы обнаружены в смесях классического однохвостого с двухвостым ПАВ, например, в смеси SDS с бромидом дидодецилдиметиламмония (DDAB) [20,21].

Важнейший этап в развитии представлений о морфологии агрегированных структур - введение фактора упаковки молекул ПАВ в работах Израелашвили и сотр. [12]. Эта величина ($P$) представляет собой отношение объема хвоста молекулы ПАВ $v$ к длине максимально растянутого хвоста $l_c$ и оптимальной площади поверхности

$a_{opt}$, приходящейся на молекулу ПАВ в агрегате: $P = {}^{v}/_{a_{opt} l_C}$. Величина $a_{opt}$ определяется, как площадь на одну молекулу ПАВ при минимальной свободной энергии мицеллообразования. Для сферических мицелл фактор упаковки равен 1/3, для цилиндрических – 1/2 и для плоских бислоев – 1. Для обратных мицелл P > 1. Израелашвили показал, что образование везикул с оптимальным значением площади на молекулу $a_{opt}$ сопровождается упаковкой молекул ПАВ в усеченный конус; в таком случае фактор упаковки 1/2 < P < 1. Это накладывает ограничения на минимальный (критический) радиус везикулы. При сильном искривлении поверхности сферического бислоя для сохранения благоприятного значения фактора упаковки необходимо увеличивать толщину внешнего слоя, которая не может быть больше значения $l_C$. Таким образом, Израелашвили получено приближенное значение для критического внешнего радиуса везикулы (ур. 1):

$$R_C \approx \frac{l_C}{1 - {}^{v}/_{a_{opt} l_C}} \qquad (1)$$

В настоящее время имеется хорошо разработанный аппарат феноменологической термодинамики для описания агрегации в растворах ПАВ [22]. Предсказывать образование агрегатов в растворах ПАВ, исходя из молекулярных параметров амфифилов, позволяют молекулярно-термодинамические модели. Наиболее известными являются модели Нагаражана и Рункенштейна [9], Бланктшайна и сотр. [10], Нинхэма и сотр. [8,23]. Согласно этим теориям, существует несколько вкладов в свободную энергию образования мицелл, каждый из которых можно описать своим набором формул. Более подробно эти вклады будут рассмотрены в разделе 2.2. Каждая из упомянутых моделей содержит целый ряд приближений, поэтому для увеличения предсказательной способности моделей необходимо их усовершенствование. В частности, требуется более детальный подход к решению электростатической задачи, классическое описание которой приводится в разделе 2.3.

Поиск минимума свободной энергии мицеллообразования, которая вычисляется согласно выбранной модели, позволяет найти энергетически стабильные агрегаты.

### 2.2. Классическая модель Нагаражана-Рукенштейна

Нами рассматриваются катанионные везикулы, состоящие из ПАВ вида А и В. Расчет свободной энергии агрегации подразумевает использование геометрических характеристик, таких как объем агрегата и площадь поверхности гидрофобно-гидрофильной границы. Их определение при моделировании смешанных везикул означает нахождение следующих оптимальных параметров: внутреннего радиуса везикулы $R$, толщин внешнего и внутреннего слоев $\tau_{out}$ и $\tau_{in}$, составов внешнего и внутреннего слоев $\alpha_{out} = \frac{N_{A\,out}}{N_{A\,out}+N_{B\,out}}$ и $\alpha_{in} = \frac{N_{A\,in}}{N_{A\,in}+N_{B\,in}}$, где $N_{A\,out}$ и $N_{A\,in}$ – число молекул ПАВ вида А во внешнем и внутреннем слоях, $N_{B\,out}$ и $N_{B\,in}$ - число молекул ПАВ вида В во внешнем и внутреннем слоях везикулы соответственно.

Согласно модели Нагаражана-Рукенштейна [9], свободную энергию мицеллообразования можно представить как сумму нескольких вкладов (ур. 2):

$$g = g_{tr} + g_{int} + g_{rep} + g_{def} + g_{ion} + g_{mix} \qquad (2),$$

где $g_{tr}$ – гидрофобный вклад, $g_{int}$ – вклад поверхностного натяжения межфазной границы, $g_{rep}$ – стерический вклад отталкивания голов ПАВ, $g_{def}$ – деформационный вклад, $g_{ion}$ – электростатический вклад, который учитывается в случае наличия ионных ПАВ, $g_{mix}$ – вклад смешения, он присутствует, если в системе находится смесь нескольких ПАВ.

Гидрофобный вклад является основной движущей силой агрегации молекул ПАВ в растворе. Это связано с плохой растворимостью углеводородных хвостов в воде и их стремлением избежать соприкосновения с ней. Для каждой группы $CH_3$-, $CH_2$-, CH-, согласно модели, существует своя эмпирическая формула для расчета данного вклада в зависимости от температуры [9]. Гидрофобный вклад, приходящийся на одну молекулу ПАВ, рассчитывается исходя из количества каждой из групп в углеводородном хвосте данной молекулы.

Вклад, отвечающий за поверхностное натяжение границы «углеводород – вода», определяется приходящейся на одну молекулу ПАВ площадью поверхности гидрофобно-гидрофильной границы, и площадью экранирования полярной головой

ПАВ углеводородного хвоста от контакта с водой. Та часть поверхности, которая остается неэкранированной, и определяет величину данного вклада (ур. 3).

$$g_{int} = \sigma_{S-W}(a - a_0) \quad (3),$$

где $\sigma_{S-W}$ – межфазное натяжение границы углеводород-вода, $a$ – площадь поверхности, приходящаяся на одну молекулу ПАВ, $a_0$ – площадь экранирования межфазной границы головой ПАВ.

Вклад стерического отталкивания голов ПАВ отражает взаимодействия исключенного объема между головами ПАВ, находящимися, согласно классическим моделям, на фиксированном расстоянии относительно гидрофобно-гидрофильной границы. Обычно этот вклад оценивается на основе того или иного двумерного уравнения состояния [9] и определяется отношением молекулярной площади сечения головы ПАВ ($a_p$) к площади поверхности, которая приходится на одну молекулу ПАВ в слое ($a$). В модели Нагаражана-Рукенштейна используется двумерное уравнение состояния Ван-дер-Ваальса (ур. 4):

$$g_{rep} = -\ln(1 - \frac{a_p}{a}) \quad (4)$$

Деформационный вклад зависит от того, насколько сильно растянут углеводородный хвост в данном слое везикулы. Для расчета энергии деформации используется теория Семенова для блоксополимеров [24], исходя из которой данный вклад имеет квадратичную зависимость от толщины слоя везикулы, в котором находится рассматриваемая молекула ПАВ, а численное значение коэффициента определяется геометрией агрегата [9].

Вклад смешения хвостов ПАВ в ядре оценивается с помощью теории Флори-Хаггинса. В классическом варианте модели параметры Флори выражены через параметры растворимости Гильдебранда, величины которых известны для широкого круга веществ [25].

При рассмотрении везикул стоит обратить внимание на то, что они имеют форму сферического бислоя, а значит, имеют два слоя (внутренний и внешний). Таким образом, при расчете свободной энергии образования везикулы на одну

молекулу ПАВ необходимо использовать параметры, которые учитывают относительные вклады слоев в величину свободной энергии: $\chi_{out} = \frac{N_{out}}{N_{out}+N_{in}}$ и $\chi_{in} = \frac{N_{in}}{N_{out}+N_{in}}$, где $N_{out}$ и $N_{in}$ – число молекул ПАВ во внешнем и внутреннем слоях везикулы соответственно. В таком случае свободная энергия агрегации примет следующий вид (ур. 5):

$$g = \chi_{out} g_{out} + \chi_{in} g_{in} \quad (5),$$

где $g_{out}$ и $g_{in}$ – свободные энергии агрегации внешнего и внутреннего слоев соответственно.

Вычисление электростатического вклада, обусловленного тем, что головы ПАВ заряжены, рассматривается нами в следующем разделе 2.3.

### 2.3. Электростатический вклад в свободную энергию

Расчет электростатического вклада в свободную энергию ведется согласно теории Гуи-Чапмена, в которой учитывается возникновение двойного электрического слоя вблизи заряженной поверхности. Для расчета электростатического потенциала используют нелинеаризованное уравнение Пуассона-Больцмана (НЛПБ, ур. 6):

$$\vec{\nabla}\left(-\varepsilon\vec{\nabla}\varphi\right) = \sum_i C_i^{bulk} z_i e_0 e^{-\frac{z_i \varphi e_0}{k_B T}} \quad (6),$$

где $\varphi$ – значение потенциала в данной точке пространства, $\varepsilon$ – диэлектрическая проницаемость среды, $e_0$ – элементарный заряд, $C_i^{bulk}$ - концентрация иона сорта $i$ при бесконечном отдалении от заряженной поверхности, $z_i$ – зарядовое число иона сорта $i$, $k_B$ – константа Больцмана, $T$ – температура.

Для расчета электростатического вклада в свободную энергию необходимо провести интегрирование потенциала по величине поверхностного заряда мицеллы $q_{mic}$. Такое интегрирование по физическому смыслу представляет собой процесс заряжения поверхности агрегата от нуля до значения $q_{mic}$ (ур. 7):

$$G_{ion} = \int_0^{q_{mic}} \varphi dq \quad (7)$$

Стоит отметить, что уравнение НЛПБ не имеет аналитического решения для сферической симметрии, поэтому для вычисления электростатической энергии везикулы применяют численные методы или используют линеаризованное уравнение Пуассона-Больцмана (ЛПБ, ур. 8) при условии небольших значений потенциалов $|\varphi| \ll \frac{k_B T}{e}$.

$$\frac{d^2\varphi(r)}{dr^2} + \frac{2}{r}\frac{d\varphi(r)}{dr} - \varphi(r) = 0 \qquad (8)$$

В большинстве работ используется приближение электронейтральности внутренней части везикулы. Для вычисления электростатического вклада в свободную энергию Нагаражан [9] использует приближенное решение уравнения Пуассона-Больцмана для плоской геометрии с добавлением фактора кривизны в случае рассмотрения везикул и других искривленных морфологий. Винтерхальтер [26] для расчета электростатической энергии использует теорию ЛПБ и разложение по кривизне.

В работах Бланкштайна [27] и Брашера [17] рассматривался общий случай возможной регуляции заряда внутри и снаружи везикулы в рамках теории НЛПБ. Электростатический вклад в свободную энергию рассчитывался путем численного решения интегрального уравнения. Однако в данных работах авторы не вводили понятие трансмембранного потенциала и не рассматривали его поведение в различных условиях. Поэтому в настоящем исследовании этим задачам будет уделено особое внимание.

### 2.4. Агрегативное равновесие и материальный баланс

Для учета нахождения в системе везикул разного размера необходимо построение сложных вычислительных алгоритмов, что затрудняет поиск решения данной задачи, поэтому чаще всего используется приближение о монодисперсности везикул [9,10].

Нагаражан [9] рассматривал зависимость числа агрегации везикулы от ее состава для системы, состоящей из катионного и анионного ПАВ. Бланктшайн [10]

так же с учетом агрегативного равновесия исследовал зависимость концентрации везикулы от ее состава и числа агрегации. Однако в данных работах не была изучена возможность сосуществования везикул разного размера при заданной брутто-концентрации ПАВ. Для везикул постановка такой задачи наиболее актуальна, поскольку для них характерна большая полидисперсность.

Для определения стабильных агрегатов, помимо нахождения минимума свободной энергии агрегации, необходимо учитывать энтропийный фактор. Это требует рассмотрения равновесия между мицеллой и мономерными неагрегированными ПАВ. Химический потенциал мицеллы, образованной из $N_A$ молекул ПАВ вида А и $N_B$ молекул ПАВ вида В, можно записать следующим образом (ур. 9):

$$\mu_N = N_A\mu_{1A} + N_B\mu_{1B} \quad (9),$$

где $\mu_N$ – химический потенциал мицеллы, состоящей из $N = N_A + N_B$ молекул ПАВ, $\mu_{1A}$ и $\mu_{1B}$ - химические потенциалы мономеров ПАВ вида А и В соответственно.

Для разбавленного раствора химический потенциал компонента системы имеет вид (ур. 10):

$$\mu = \mu^* + kTlnX \quad (10),$$

где $\mu^*$ – стандартный химический потенциал компонента, $X$ - мольная доля компонента в растворе.

Используя (ур. 10) для химических потенциалов $\mu_N$, $\mu_{1A}$ и $\mu_{1B}$ совместно с (ур. 9), получаем (ур. 11):

$$X_N = X_{1A}^{N_A} \cdot X_{1B}^{N_B} \cdot e^{-N\cdot\Delta\mu_N^*} \quad (11),$$

где $X_N$ – мольная доля мицеллы с числом агрегации $N$, $X_{1A}$ – мольная доля мономеров ПАВ вида А, $X_{1B}$ – мольная доля мономеров ПАВ вида В, $\Delta\mu_N^* = \frac{\mu_N^* - N_A\cdot\mu_{1A}^* - N_B\cdot\mu_{1B}^*}{N}$ - стандартная свободная энергия мицеллообразования.

Как видно из данной формулы (ур. 11), увеличение числа агрегации вызывает заметное снижение равновесной концентрации $X_N$, а уменьшение $\Delta\mu_N^*$ способствует ее росту.

Обозначим исходные брутто-концентрации ПАВ вида А и В: $X_{tot\,A}$ и $X_{tot\,B}$. Общая брутто-концентрация ПАВ является их суммой: $X_{tot} = X_{tot\,A} + X_{tot\,B}$. Также введем величину концентрации $X_1$ для мономеров неагрегированных ПАВ как сумму концентраций мономеров ПАВ вида А и В: $X_1 = X_{1A} + X_{1B}$.

Зная исходную концентрацию каждого вида ПАВ в растворе, можно записать уравнение материального баланса в общем виде (ур. 12):

$$X_{tot\,A} + X_{tot\,B} = X_{1A} + X_{1B} + \sum_m \sum_{N_m} N_m \cdot X_{N_m} \quad (12)$$

Данное выражение отражает тот факт, что ПАВ в растворе находится либо в мономерной форме, либо в составе агрегата. Расчет агрегированного количества ПАВ подразумевает суммирование по агрегатам всех форм ($m$) и размеров ($N_m$). При этом каждое значение $X_{N_m}$ удовлетворяет агрегативному равновесию (ур. 11). Процесс решения системы из $m \cdot N_m$ уравнений (ур. 11) и уравнения (ур. 12) является сложной вычислительной задачей, которой посвящена значительная часть настоящей работы.

Таким образом, в данной работе были поставлены следующие задачи:

- решение уравнения ЛПБ для везикулы с учетом возможного нарушения электронейтральности её внутренней части, получение аналитического выражения для электростатической энергии образования везикулы;

- получение аналитического решения для трансмембранного потенциала в приближении ЛПБ и анализ его поведения при изменении параметров системы: солености раствора, диэлектрической проницаемости мембраны, радиуса, толщины и составов слоев везикулы, молекулярных параметров ПАВ; сопоставление полученных данных с результатами численного решения НЛПБ;

- разработка программного обеспечения для молекулярно-термодинамического моделирования свободной энергии мицеллообразования везикул;

- разработка алгоритмов и написание программного обеспечения для моделирования агрегативного равновесия в растворе смешанных ПАВ с учетом уравнений материального баланса;

- установление образования стабильных везикул и прогнозирование их структурных характеристик и равновесного распределения по составу и размерам в зависимости от концентрации ПАВ и солевого фона;

- проведение расчетов с помощью разработанного программного обеспечения с целью апробации модели для смесей классических ПАВ ($C_{16}$TAB+SOS, DTAB+SDS, $C_{16}$mimCl+SDBS) и сопоставление с имеющимися экспериментальными данными [4,16,19].

## 3. Теоретическая часть

### 3.1. Трансмембранный потенциал и электростатический вклад в свободную энергию

В данной работе мы рассматриваем образование катанионных везикул, поверхностные слои которых в большинстве наблюдаемых случаев заряжены слабо. По этой причине становится целесообразным использование теории ЛПБ для описания электростатических взаимодействий.

Схематичное двумерное изображение везикулы представлено на рис. 1.

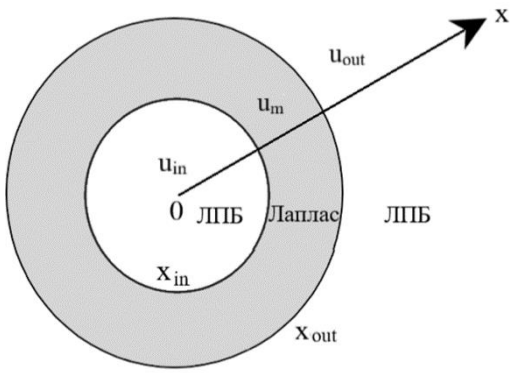


**Рис. 1.** Двумерное изображение везикулы с указанием внешнего, мембранного и внутреннего потенциалов, а также областей действия уравнений ЛПБ и Лапласа.

Для удобства введем безразмерные величины потенциала $u(x) = \frac{e_0\varphi(r)}{k_BT}$, координаты $x = \kappa \cdot r$ и поверхностной плотности заряда $t = \frac{qe_0}{a\kappa\varepsilon_0 k_BT}$, где $\kappa$ - обратная длина Дебая.

Для получения выражений для внешнего $u_{out}$ и внутреннего $u_{in}$ потенциалов воспользуемся решением ЛПБ для сферической геометрии с константами A и B (ур. 13):

$$u(x) = \frac{Ae^{-x}}{x} + \frac{Be^{x}}{x} \qquad (13)$$

В силу отсутствия подвижных зарядов внутри углеводородного слоя, мембранный потенциал $u_m$ подчиняется уравнению Лапласа (ур. 14):

$$\frac{d}{dx}(\frac{x^2 du_m}{dx}) = 0 \quad (14)$$

Интегрирование выражения (ур. 14) приводит к (ур. 15):

$$\frac{du_m}{dx} = \frac{1}{x^2}\frac{x_{in}x_{out}}{x_{in} - x_{out}}\Delta_m \quad (15),$$

где $\Delta_m = u_m(x_{in}) - u_m(x_{out})$ – трансмембранный потенциал, $x_{out}$ – координата поверхности внешнего слоя, $x_{in}$ – координата поверхности внутреннего слоя.

Согласно закону Гаусса, скачки напряженности на внешней и внутренней заряженных поверхностях везикулы (ур. 16, 17) пропорциональны соответствующим поверхностным плотностям заряда $t_{out}$ и $t_{in}$:

$$\varepsilon_m\left(\frac{du_m}{dx}\right)_{x=x_{out}} - \varepsilon_w\left(\frac{du_{out}}{dx}\right)_{x=x_{out}} = t_{out} \quad (16)$$

$$\varepsilon_w\left(\frac{du_{in}}{dx}\right)_{x=x_{in}} - \varepsilon_m\left(\frac{du_m}{dx}\right)_{x=x_{in}} = t_{in} \quad (17),$$

где $\varepsilon_m$ – диэлектрическая проницаемость углеводородного слоя мембраны, 2.1, $\varepsilon_w$ – диэлектрическая проницаемость воды, 78.2.

Подставляя (ур. 15) в уравнения (ур. 16) и (ур. 17), получаем граничные условия (ур. 18, 19), записанные через эффективные поверхностные зарядовые плотности $t_{out}^{eff}$ и $t_{in}^{eff}$:

$$\left(\frac{du_{out}}{dx}\right)_{x=x_{out}+0} = -t_{out}^{eff}(\Delta_m) \quad (18)$$

$$\left(\frac{du_{in}}{dx}\right)_{x=x_{in}-0} = t_{in}^{eff}(\Delta_m) \quad (19)$$

При этом выражения для эффективных поверхностных плотностей заряда имеют следующий вид (ур. 20, 21):

$$t_{out}^{eff}(\Delta_m) = \frac{t_{out}}{\varepsilon_w} - \frac{\varepsilon_m}{\varepsilon_w}\frac{x_{in}}{x_{out}}\frac{\Delta_m}{(x_{in} - x_{out})} \quad (20)$$

$$t_{in}^{eff}(\Delta_m) = \frac{t_{in}}{\varepsilon_w} + \frac{\varepsilon_m}{\varepsilon_w}\frac{x_{out}}{x_{in}}\frac{\Delta_m}{(x_{in} - x_{out})} \quad (21)$$

Для дальнейших преобразований необходимо использование следующих граничных условий: при бесконечном отдалении от везикулы и в ее центре производные потенциалов обращаются в ноль (ур. 22, 23):

$$\left.\frac{du_{out}}{dx}\right|_{x=\infty} = 0 \quad (22)$$

$$\left.\frac{du_{in}}{dx}\right|_{x=0} = 0 \quad (23)$$

Решение ЛПБ (ур. 13) для внешнего и внутреннего потенциалов везикулы с учетом (ур. 18, 19, 22, 23) приводит к следующим выражением (ур. 24, 25):

$$u_{out}(x) = t_{out}^{eff}(\Delta_m)\frac{x_{out}^2}{1 + x_{out}}\frac{e^{(x_{out}-x)}}{x} \quad (24)$$

$$u_{in}(x) = t_{in}^{eff}(\Delta_m)\frac{x_{in}^2}{[x_{in}\cosh(x_{in}) - \sinh(x_{in})]}\frac{\sinh(x)}{x} \quad (25)$$

Из уравнений (ур. 24) и (ур. 25) следует, что потенциалы имеют линейную зависимость от трансмембранного потенциала, который, в свою очередь, зависит от значений потенциалов на заряженных поверхностях. Благодаря этому, нахождение трансмембранного потенциала сводится к подстановке выражений (ур. 24, 25) для $u_{in}(x_{in})$ и $u_{out}(x_{out})$ в определение $\Delta_m$ с учетом непрерывности потенциалов на границах заряженных поверхностей: $u_m(x_{in}) = u_{in}(x_{in})$, $u_m(x_{out}) = u_{out}(x_{out})$.

Полученное аналитическое выражение для трансмембранного потенциала (ур. 26) содержит зависимость от поверхностных плотностей заряда, внешнего и внутреннего радиуса везикулы, диэлектрической проницаемости:

$$\Delta_{\mathrm{m}} = \frac{t_{in}\frac{x_{in}}{x_{in}\mathrm{cth}(x_{in})\text{-}1} - t_{out}\frac{x_{out}}{1+x_{out}}}{\varepsilon_w - \frac{\varepsilon_{\mathrm{m}}}{(x_{in} - \mathrm{x_{out}})}\left[\frac{x_{out}}{x_{in}\mathrm{cth}(x_{in})\text{-}1} + \frac{x_{in}}{1+x_{out}}\right]} \quad (26)$$

Зависимость трансмембранного потенциала от солевого фона и кислотности водного раствора отражается его зависимостью от обратной длины Дебая [м$^{-1}$] (ур. 27):

$$\kappa = \sqrt{\frac{4\pi}{k_B T} e_0{}^2 N_A 1000 \sum C_i^{bulk} z_i^2} \quad (27),$$

где $N_A$ – число Авогадро, $C_i^{bulk}$ указывается в [моль/л].

Разложение уравнения (ур. 26) по кривизне дает следующее выражение (ур. 28):

$$\Delta_m = \frac{t_{in} - t_{out}}{\varepsilon_{\mathrm{w}} + 2\varepsilon_{\mathrm{m}}\tau^{-1}} + \frac{t_{in} + t_{out}}{\varepsilon_{\mathrm{w}} + 2\varepsilon_{\mathrm{m}}\tau^{-1}}\frac{1}{R} + O\left(\frac{1}{R^2}\right) \quad (28),$$

где $R$ – внутренний радиус везикулы, $\tau$ – толщина мембраны.

Используя уравнение (ур. 28) для везикулы бесконечного радиуса, получаем выражение для трансмембранного потенциала асимметрично заряженного плоского бислоя (ур. 29):

$$\Delta_m^{plane} = \frac{t_{in} - t_{out}}{\varepsilon_{\mathrm{w}} + 2\varepsilon_{\mathrm{m}}\tau^{-1}} \quad (29)$$

Поскольку в рамках теории ЛПБ электростатический потенциал является линейной функцией поверхностных плотностей заряда слоев мембраны, наиболее простым способом получения электростатической свободной энергии везикулы является расчет работы заряжения ее поверхностей [28] (SI), что дает следующее выражение (ур. 30):

$$g_{el} = \chi_{out}\frac{\kappa a_{out}}{8\pi l_B} t_{out} u_{out}(x_{out}) + \chi_{in}\frac{\kappa a_{in}}{8\pi l_B} t_{in} u_{in}(x_{in}) \quad (30),$$

где $a_{out}$ и $a_{in}$ – средние площади на одну молекулу ПАВ внешней и внутренней поверхности гидрофобно-гидрофильной границы соответственно, $l_B = \frac{{e_0}^2}{4\pi\varepsilon_w\varepsilon_0 k_B T}$ - длина Бьеррума.

Выражение (ур. 30) учитывает поляризацию диэлектрической мембраны в электрическом поле [28] (SI). При этом для углеводородных мембран в везикулах вклад поляризации в общую электростатическую энергию невелик. Так, например, для везикулы CTAB+SOS с параметрами $R = 5l_{C(SOS)}$, $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\, l_{C(SOS)}$, $C_{salt} = 100$ мМ, где $l_{C(SOS)}$ – длина максимально растянутого хвоста SOS, вклад поляризации мембраны составляет лишь сотую долю процента от общей электростатической свободной энергии ($g_{el}$ = 7.3 $10^{-1}$).

### 3.2. Алгоритм моделирования везикул в катанионных растворах ПАВ

Как и в разделе 2.4, рассмотрим раствор двух ПАВ, одно из которых катионное (А), другое – анионное (В). Предполагаем, что в растворе присутствуют монодисперсные сферические мицеллы, полидисперсные стержнеобразные мицеллы и полидисперсные везикулы. Уравнение материального баланса (ур. 12) записывается следующим образом (ур. 31):

$$X_{tot\,A} + X_{tot\,B} = X_{1A} + X_{1B} + N_{sph}X_{N_{sph}} + \sum_{N_{cyl}} N_{cyl}X_{N_{cyl}} + \sum_{N_{ves}} N_{ves}X_{N_{ves}} \quad (31)$$

Поиск решения уравнения (ур. 31) требует составления сложного вычислительного алгоритма, поскольку для разных соотношений $X_{1A}$ и $X_{1B}$ необходим отдельный расчет свободных энергий агрегаций мицелл и их концентраций. Чтобы облегчить задачу, будем задавать долю мономерного ПАВ А $\alpha_{A1} = \frac{X_{1A}}{X_1}$. Соответственно, для ПАВ В $\alpha_{B1} = \frac{X_{1B}}{X_1} = 1 - \alpha_{A1}$. При заданной мономерной доле решение уравнения (ур. 31) сводится к одномерному поиску суммарной мольной доли мономеров: $X_1 = X_{1A} + X_{1B}$.

В данной работе использовался следующий алгоритм (рис. 2).

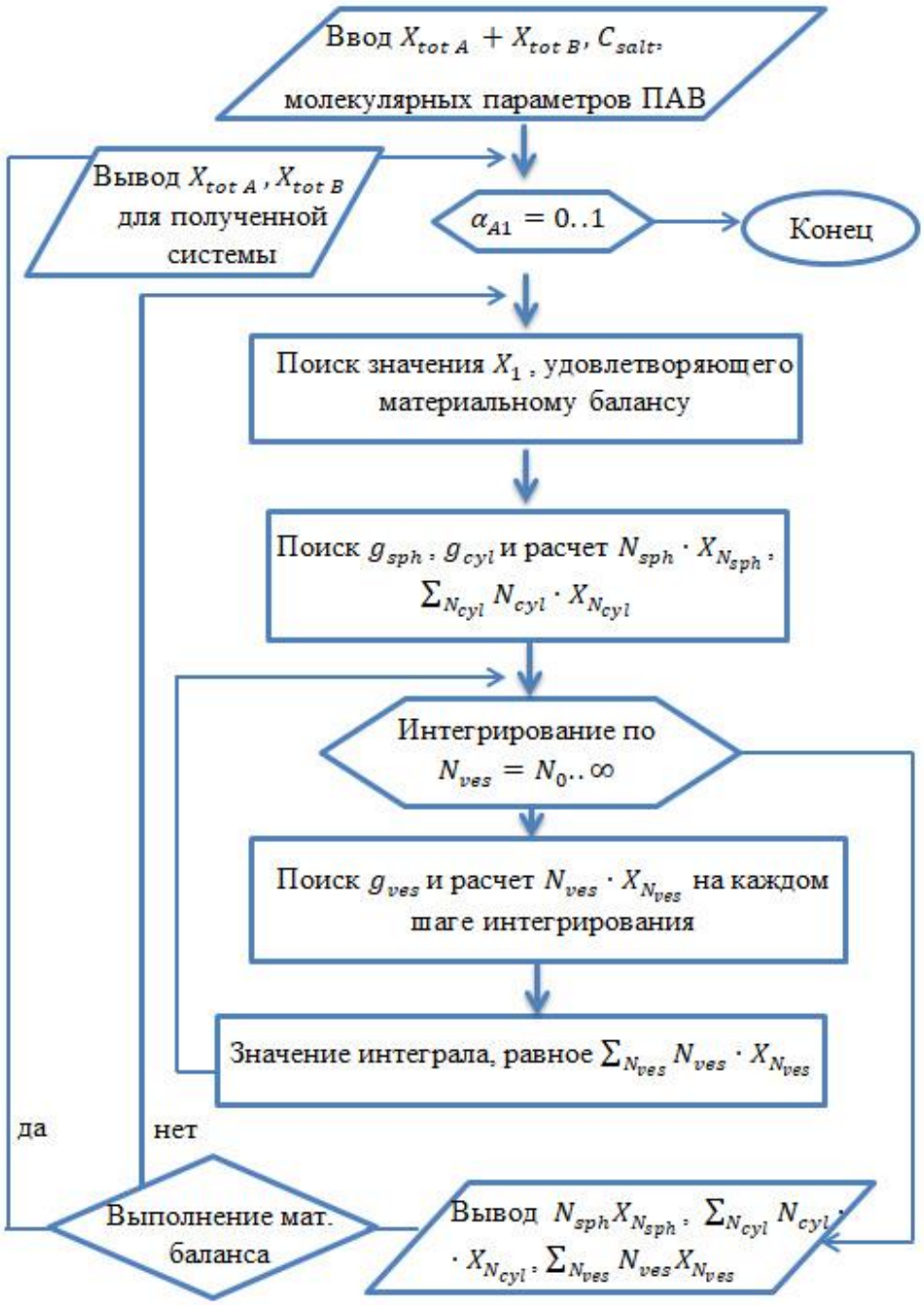


**Рис. 2.** Вычислительный алгоритм программы для поиска стабильных агрегатов и их концентрации с учетом материального баланса.

Входными параметрами для выполнения расчетов являются общая брутто-концентрация ПАВ $X_{tot\,A} + X_{tot\,B}$, концентрация фонового 1:1-электролита $C_{salt}$ и молекулярные параметры обоих ПАВ: число углеродов $n_C$ в алкильном углеводородном хвосте и площадь поперечного сечения головы $a_p$.

На первом этапе расчетов задается некоторое значение $\alpha_{A1}$ в диапазоне значений от 0 до 1 и методом дихотомии ведется одномерный поиск величины $X_1$, которая будет соответствовать выполнению уравнения материального баланса (ур. 31) с заданной точностью. Для того чтобы проверить выполнение материального баланса, необходимо рассчитать концентрации всех агрегатов в системе согласно агрегативному равновесию (ур. 11).

Для сферических мицелл происходит поиск такого значения свободной энергии мицеллообразования сфер $g_{sph}$, которое соответствует максимальной концентрации $X_{N_{sph}}$ согласно (ур. 11). Нахождение $g_{sph}$ происходит при варьировании радиуса сферы и ее состава методом золотого сечения.

Расчет распределения концентраций стержнеобразных мицелл $\sum_{N_{cyl}} N_{cyl} X_{N_{cyl}}$ ведется согласно аналитической формуле [29]. Сначала идет поиск оптимального радиуса и состава, соответствующих бесконечному цилиндру, затем рассчитывается оптимальный размер и состав сферического окончания цилиндра. С использованием найденных оптимальных значений радиусов и составов проводится аналитическое суммирование всей популяции цилиндров от минимального до бесконечно большого значения числа агрегации. Указанное суммирование сводится к нахождению суммы геометрической прогрессии [29].

Для полидисперсной популяции везикул мольные доли агрегированного ПАВ $\sum_{N_{ves}} N_{ves} X_{N_{ves}}$ определяется численным интегрированием по числу агрегации везикулы $N_{ves}$ от некоторого начального значения $N_0$ до бесконечности (ур. 32):

$$\sum_{N_{ves}} N_{ves} X_{N_{ves}} \cong \int_{N_0}^{\infty} N_{ves} X_{N_{ves}} dN_{ves} \qquad (32)$$

Интегрирование проводили методом трапеций.

Алгоритм расчета свободной энергии агрегации $g_{ves}$ следующий: для заданного на текущем шаге интегрирования внутреннего радиуса везикулы $R$ происходит поиск оптимальной свободной энергии по остальным параметрам агрегата. Составы внешнего и внутреннего слоев везикулы $\alpha_{out}$ и $\alpha_{in}$ перебираются в некотором диапазоне значений. При заданных значениях составов свободная энергия

$g_{ves}$, соответствующая максимальной концентрации везикул, определяется с помощью симплекс метода относительно варьирования общей толщины углеводородного слоя $\tau = \tau_{out} + \tau_{in}$ при постоянстве соотношения толщин слоев $\tau_{in}/\tau_{out}$. Оптимальные значения $\alpha_{out}$, $\alpha_{in}$ выбираются путем сравнения концентраций везикул для каждой комбинации составов.

После расчета оптимальной $g_{ves}$ радиусу $R$ ставится в соответствие $N_{ves}$ и вычисляется $N_{ves} \cdot X_{N_{ves}}$ согласно (ур. 11).

После определения концентрации агрегированных ПАВ проверяется выполнение уравнения материального баланса. Если равенство (ур. 31) не подтверждается, алгоритм присваивает новое значение $X_1$ согласно методу дихотомии. Если условие выполняется, то полученные данные сохраняются, и происходит переход к следующему значению $\alpha_{A1}$. В результате расчетов для каждого значения $\alpha_{A1}$ выводятся индивидуальные брутто-концентрации ПАВ $X_{tot\,A}$ и $X_{tot\,B}$, характеристики агрегатов, их распределение по размерам и оптимальные значения свободной энергии агрегации. В программе также предусмотрен расчет критической концентрации мицеллообразования (ККМ): вычисления проводятся исходя из равенства количества агрегированного и неагрегированного ПАВ.

Описанный алгоритм реализован в программном обеспечении, написанном на языке FORTRAN (Приложение А).

## 4. Обсуждение результатов

### 4.1. Влияние геометрических характеристик везикулы и солевого фона на трансмембранный потенциал

Для иллюстрации поведения трансмембранного потенциала рассмотрим модельную систему, состоящую из симметричных катионного и анионного ПАВ, где количество углеродов в хвосте $n_{C1} = n_{C2} = 8$, площадь поперечного сечения головы $a_{p1} = a_{p2} = 17$ Å$^2$. Данное значение $a_p$ выбрано равным величине для алкилсульфатов [9]. Толщины слоев везикулы и внутренний радиус будем выражать в единицах максимально растянутого углеводородного хвоста $l_C$. Зависимости для трансмембранного потенциала, рассчитанные по (ур. 26) теории ЛПБ, сопоставляются с данными, полученными в результате численного решения НЛПБ.

Рассмотрим везикулу, в обоих слоях которой превалируют катионные ПАВ: $\alpha_{in} = 0.52$, $\alpha_{out} = 0.56$, толщины слоев одинаковы $\tau_{in} = \tau_{out} = 0.6\, l_C$. Зависимость трансмембранного потенциала от радиуса везикулы при различном солевом фоне раствора проиллюстрирована на рис. 3.

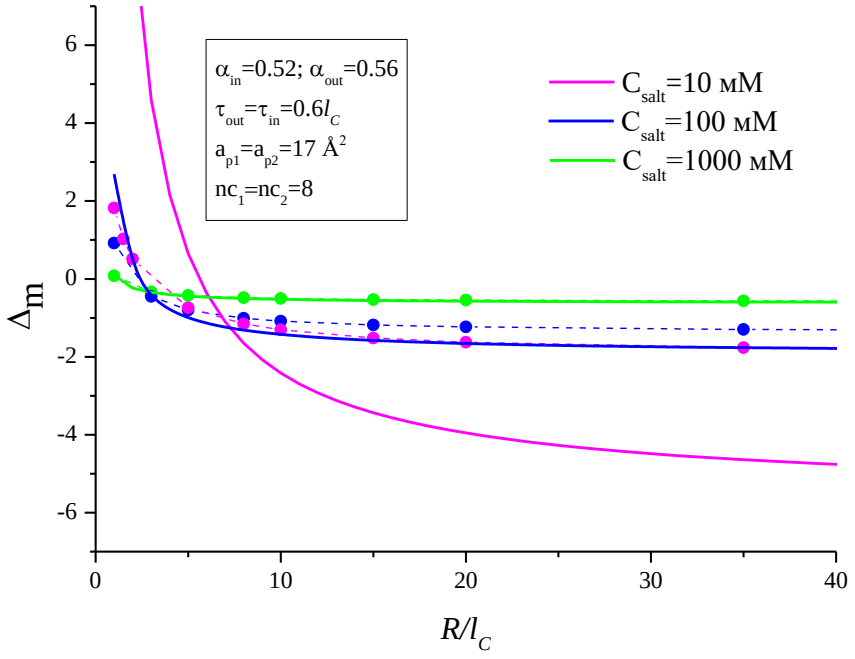


**Рис. 3.** Зависимость трансмембранного потенциала от внутреннего радиуса везикулы для системы $C_8$+$C_8$ с $a_{p1} = a_{p2} = 17$ Å$^2$ в растворе, содержащем 1000 мМ, 100 мМ, и 10 мМ соли. Сплошные линии: ЛПБ, пунктир + символы: НЛПБ.

Для небольших рассматриваемых на рис. 3 везикул трансмембранный потенциал положителен, так как потенциал на внутренней стороне мембраны больше, чем на внешней, несмотря на то, что в данном случае внешний слой заряжен несколько сильнее ($\alpha_{out} > \alpha_{in}$). Сильно искривленная внутренняя поверхность мембраны является причиной небольшого значения площади поверхности $a_{in}$, приходящейся на одну молекулу ПАВ, и, как следствие, увеличения поверхностной плотности заряда $t_{in}$, что, согласно (ур. 25) вызывает значительное повышение потенциала $u_{in}$. Увеличение радиуса везикулы уменьшает кривизну мембраны: площадь поверхности на молекулу увеличивается на ее внутренней стороне и уменьшается на ее внешней стороне. Плотности поверхностного заряда и изменяются противоположным образом, что приводит к уменьшению $u_{in}$ и увеличению $u_{out}$. В результате трансмембранный потенциал уменьшается, проходя через ноль с увеличением размера везикулы. Изменение знака трансмембранного потенциала означает изменение тенденции везикулы накапливать подвижные катионы и анионы в своей внутренней части. В пределе больших радиусов трансмембранный потенциал приближается к значению для плоского бислоя (ур. 29).

Как видно из рис. 3, теории ЛПБ и НЛПБ дают одинаковые результаты при больших значениях солености. Наибольшее расхождение наблюдается при низкой соли (10 мМ), поскольку слабое экранирование электростатических взаимодействий способствует росту абсолютных значений потенциалов $u_{in}$ и $u_{out}$, которые оказываются больше величины $\frac{k_B T}{e}$, что приводит к нарушению условия применимости теории ЛПБ.

На рис. 4 представлена зависимость трансмембранного потенциала от солености раствора при постоянном значении радиуса $R = 2l_C$ в случаях, когда внешний слой мембраны заряжен сильнее внутреннего: 1) $\alpha_{in} = 0.52$, $\alpha_{out} = 0.56$, 2) $\alpha_{in} = 0.52$, $\alpha_{out} = 0.58$. Толщины слоев прежние: $\tau_{in} = \tau_{out} = 0.6\, l_C$.

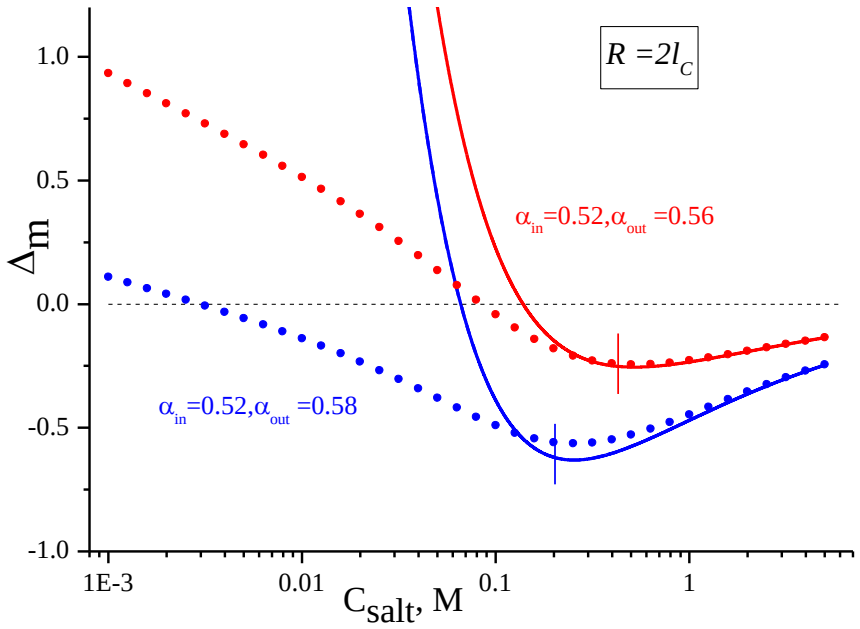


**Рис. 4.** Зависимость трансмембранного потенциала от солености раствора при постоянном значении радиуса $R = 2l_C$ и толщин $\tau_{in} = \tau_{out} = 0.6\, l_C$. Рассмотрены два случая: 1) $\alpha_{in} = 0.52$, $\alpha_{out} = 0.56$ – красный цвет, 2) $\alpha_{in} = 0.52$, $\alpha_{out} = 0.58$ – синий цвет. Сплошными линиями обозначены данные, полученные по теории ЛПБ, кружками – НЛПБ. Вертикальные засечки соответствуют положениям экстремумов, согласно формулам (ур. 33, 34).

Как показано на рис. 4, знак трансмембранного потенциала можно изменить путем добавления соли в раствор. Особый интерес вызывает нетривиальное поведение трансмембранного потенциала, имеющего минимум при определенной солености. При добавлении соли потенциалы $u_{in}$ и $u_{out}$ уменьшаются, но с разной скоростью. При низкой концентрации соли внутренний потенциал более чувствителен к изменению солености, чем внешний потенциал, при высокой концентрации соли - наоборот. При определенной солености скорости изменения внутреннего и внешнего потенциалов совпадают, что дает минимум трансмембранного потенциала.

Физическое обоснование такого поведения трансмембранного потенциала заключается в стремлении к выравниванию осмотического давления подвижных ионов в растворе и приведении внутренней части везикулы к электронейтральности. При маленькой солености раствора $C_{salt}$ заряженные слои везикулы слабо

экранированы, поэтому скачок электростатического потенциала между внутренней частью везикулы и объемом раствора велик, что приводит к обильному накоплению противоионов внутри везикулы при увеличении солености. При определенном значении солевого фона внутренняя часть везикулы станет электронейтральной, а трансмембранный потенциал – равным нулю. Однако даже при данной солености скачок потенциала может оставаться достаточно большим для того, чтобы продолжать накапливать ионы во внутреннем растворе везикулы при дальнейшем увеличении $C_{salt}$, при этом заряд внутренней части везикулы поменяет знак.

По мере накопления подвижных ионов поверхностный потенциал уменьшается и становится более экранированным, всё быстрее падая к центру везикулы. С увеличением солености во внутреннем растворе перенос противоионов будет всё меньше, а коионов – всё больше. В конце концов электрический потенциал в центре везикулы упадет до нуля, и локальная концентрация подвижных ионов достигнет объемного значения. При определенном значении солевого фона везикула содержит внутри максимальный накопленный заряд, который соответствует экстремуму трансмембранного потенциала; дальнейшее увеличение концентрации соли во внешнем растворе ведет к уменьшению накопленного везикулой избыточного мобильного заряда, возвращая внутреннюю (а следовательно, и внешнюю) часть везикулы к электронейтральности.

Поскольку экстремум наблюдается при значениях длины Дебая $\kappa^{-1} \to 0$, оценить положение минимума можно с помощью разложения (ур. 25) по $\kappa^{-1}$. Для этого вернемся к размерным величинам поверхностной плотности заряда $\sigma$, внутреннему $R$ и внешнему радиусу везикулы $R_{out}$. В результате разложения получим следующие выражения для трансмембранного потенциала и обратной длины Дебая [28] (ур. 33, 34):

$$\Delta_m \approx \left[ \left(\sigma_{in} - \sigma_{out}\right)\kappa^{-1} + \left( \frac{\sigma_{in}}{R} + \frac{\sigma_{out}}{R_{out}} \right)\kappa^{-2} \right] \frac{e_0}{\varepsilon_0 kT} \tag{33}$$

$$\kappa \approx \frac{\sigma_{in}}{\sigma_{out} - \sigma_{in}} \frac{2}{R} + \frac{\sigma_{out}}{\sigma_{out} - \sigma_{in}} \frac{2}{R_{out}} \tag{34}$$

Эти уравнения применимы в равной степени как для определения минимумов трансмембранного потенциала положительно заряженных везикул, так и для определения максимумов везикул с отрицательно заряженными мембранами.

На рис. 5 представлена зависимость трансмембранного потенциала от солености внешнего раствора для отрицательно заряженных везикул при разных значениях толщины мембраны: $\tau_{in} = \tau_{out} = 0.4\, l_C$, $\tau_{in} = \tau_{out} = 0.6\, l_C$, $\tau_{in} = \tau_{out} = 0.8\, l_C$. В одном случае внешняя поверхность мембраны заряжена сильнее внутренней $\alpha_{in} = 0.44$, $\alpha_{out} = 0.48$, во втором – наоборот, $\alpha_{in} = 0.48$, $\alpha_{out} = 0.44$.

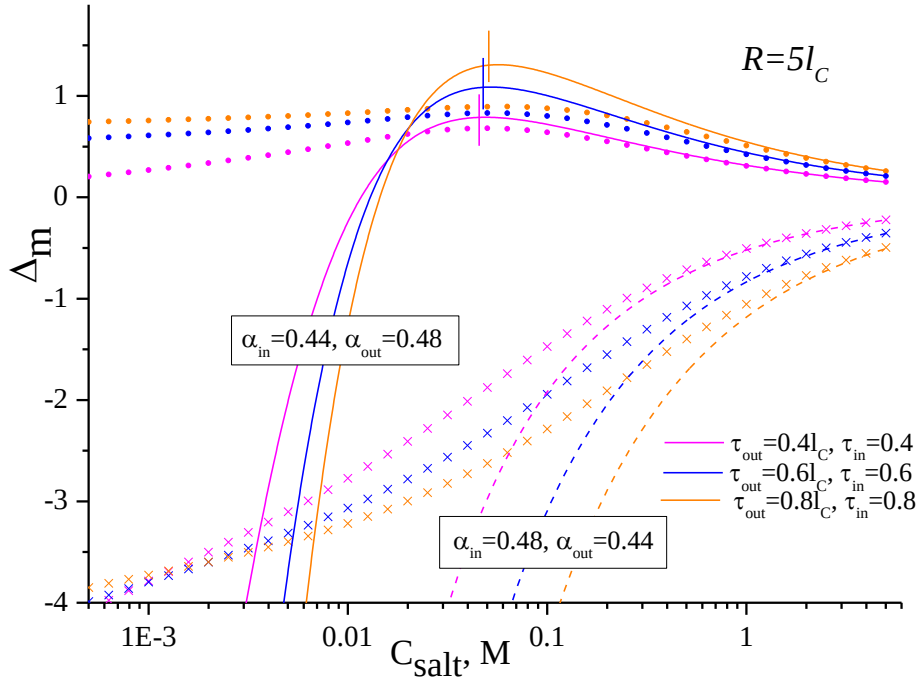


**Рис. 5.** Зависимость трансмембранного потенциала от солености внешнего раствора при постоянном значении радиуса $R = 5l_C$ и при разных значениях толщины мембраны: $\tau_{in} = \tau_{out} = 0.4\, l_C$ (розовый цвет), $\tau_{in} = \tau_{out} = 0.6\, l_C$ (синий цвет), $\tau_{in} = \tau_{out} = 0.8\, l_C$ (желтый цвет). Рассмотрены два случая: 1) $\alpha_{in} = 0.44$, $\alpha_{out} = 0.48$ – сплошные линии для ЛПБ, 2) $\alpha_{in} = 0.48$, $\alpha_{out} = 0.44$ – пунктирные линии для ЛПБ. Символами обозначается НЛПБ. Вертикальные засечки соответствуют положениям экстремумов, согласно формулам (ур. 33, 34).

Уравнение (ур. 34) показывает, что не существует экстремума, при котором внутренняя поверхность мембраны заряжена сильнее, чем внешняя поверхность $|\sigma_{in}| > |\sigma_{out}|$, поскольку $\kappa$ является положительной величиной. Это подтверждается

результатами, представленными на рис. 5: в случае, когда $|\sigma_{in}| > |\sigma_{out}|$, в зависимости трансмембранного потенциала от солености не наблюдается наличие экстремума. Для везикул, имеющих более сильную заряженную внешнюю поверхность, местоположение экстремума смещается в сторону большей солености по мере приближения $|\sigma_{out}|$ к $|\sigma_{in}|$. Согласно рис. 4 и рис. 5, положения экстремумов в теории ЛПБ и теории НЛПБ хорошо согласуются друг с другом.

Одновременное увеличение толщин слоев мембраны при постоянном внутреннем радиусе везикулы (рис. 5) приводит к увеличению поверхностной плотности заряда обоих слоев. Абсолютное значение трансмембранного потенциала увеличивается как для везикул с более заряженной внутренней поверхностью, так и для везикул с более заряженной внешней. Для везикул с составами $\alpha_{in} = 0.48$, $\alpha_{out} = 0.44$ это отражается постепенным сдвигом пунктирных кривых вниз от $\tau_{in} = \tau_{out} = 0.4\, l_C$ к $\tau_{in} = \tau_{out} = 0.8\, l_C$. Для случая $\alpha_{in} = 0.44$, $\alpha_{out} = 0.48$ увеличение значения трансмембранного потенциала проявляется сдвигом кривых вниз при низкой солености, где $\Delta_m$ отрицателен, и сдвигом вверх кривых с положительным значением $\Delta_m$ при высокой солености.

### 4.2. pH внутреннего раствора везикулы и профили электростатического потенциала

Поскольку в везикуле с небольшим радиусом электрическое поле не обращается в ноль в центре ее внутренней части, она предпочтительно накапливает либо катионы, либо анионы, в зависимости от заряда внутренней поверхности мембраны. Рассмотрим везикулу в водном растворе, содержащем HCl, NaOH и NaCl. Принимая во внимание диссоциацию молекул воды, воспользуемся выражением $K_w = [H^+][OH^-] = 10^{-14}$. Локальную концентрацию ионов внутри везикулы считаем согласно распределению Больцмана $C_\pm(x) = C_\pm^{bulk} exp(\mp u(x))$, где $C_\pm^{bulk}$ – концентрация иона при бесконечном отдалении от везикулы.

Используя распределение Больцмана для вычисления среднего значения $pH_{in}$ во внутреннем растворе везикулы, получаем формулу (ур. 35):

$$\mathrm{pH}_{in} \equiv \langle \mathrm{pH}(x) \rangle_{in} = \mathrm{pH}^{bulk} + \frac{\langle u(x) \rangle_{in}}{2.30} \qquad (35),$$

где $\langle u(x) \rangle_{in}$ – среднее значение электростатического потенциала внутри везикулы, $pH^{bulk}$ – значение $pH$ в растворе при бесконечном отдалении от везикулы.

Выражение для среднего значения потенциала рассчитывается с помощью интегрирования (ур. 25) по объему внутреннего раствора везикулы (ур. 36):

$$\langle u(x) \rangle_{in} = \frac{3}{\kappa R} t_{in}^{eff}(\Delta_m) \qquad (36)$$

На рис. 6 представлен профиль концентрации ионов водорода внутри и снаружи положительно заряженной везикулы с составами $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$ с заданным значением $pH^{bulk} = 4$.

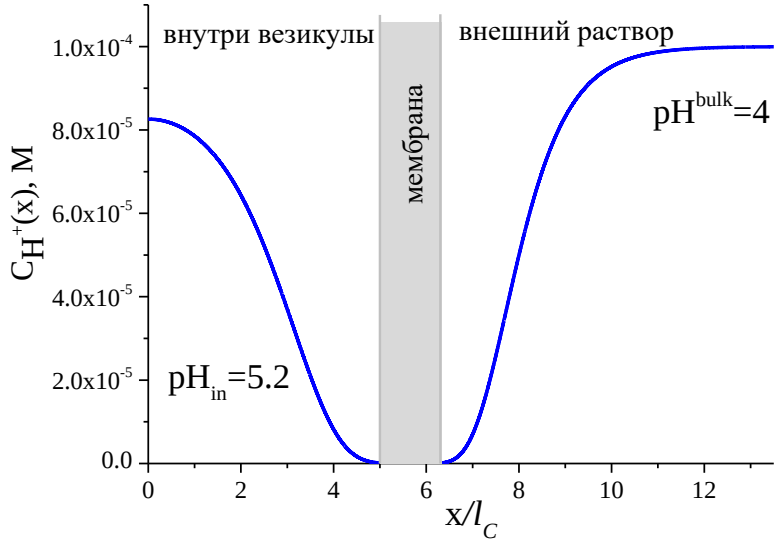


**Рис. 6.** Профиль концентрации ионов водорода внутри и снаружи везикулы $C_8$+$C_8$ с параметрами $R = 5l_C$, $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\, l_C$, $C_{salt} = 100$ мМ, $pH^{bulk} = 4$.

Положительно заряженная мембрана отталкивает ионы $H^+$ с обеих сторон и накапливает противоионы $OH^-$ и $Cl^-$ внутри везикулы, тем самым повышая $pH_{in}$ с 4 до 5.6. Вызывает интерес возможность наличия щелочного раствора внутри везикулы, находящейся в кислой среде. На рис. 7 представлена зависимость $pH_{in}$ от внутреннего радиуса везикулы с составами слоев $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$ в кислой среде

$pH^{bulk} = 4$. Согласно результатам, наличие слабощелочной среды внутри рассматриваемой везикулы возможно при значениях радиуса везикулы $R \leq 2l_C$.

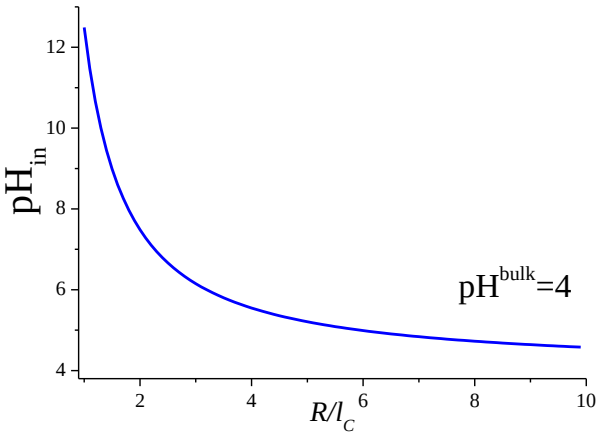


**Рис. 7.** Зависимость $pH_{in}$ от внутреннего радиуса везикулы $C_8$+$C_8$ с параметрами $R = 5l_C$, $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\, l_C$, $C_{salt} =$ 100 мМ, $pH^{bulk} = 4$.

Электростатические потенциалы и вклады в свободную энергию согласно теории НЛПБ получены при помощи разработанного нами программного обеспечения на языке FORTRAN (приложение Б). Дополнительно, расчеты потенциалов выполнены с помощью программного обеспечения COMSOL MULTIPHYSICS 4.3 с использованием 3D-электростатики и 1D-модели PDE с заданными коэффициентами. COMSOL-данные хорошо согласуются с результатами, полученными с помощью разработанного нами программного обеспечения [28] (SI). Пример распределения электростатического потенциала для везикулы в водно-солевом растворе показан на рис. 8.

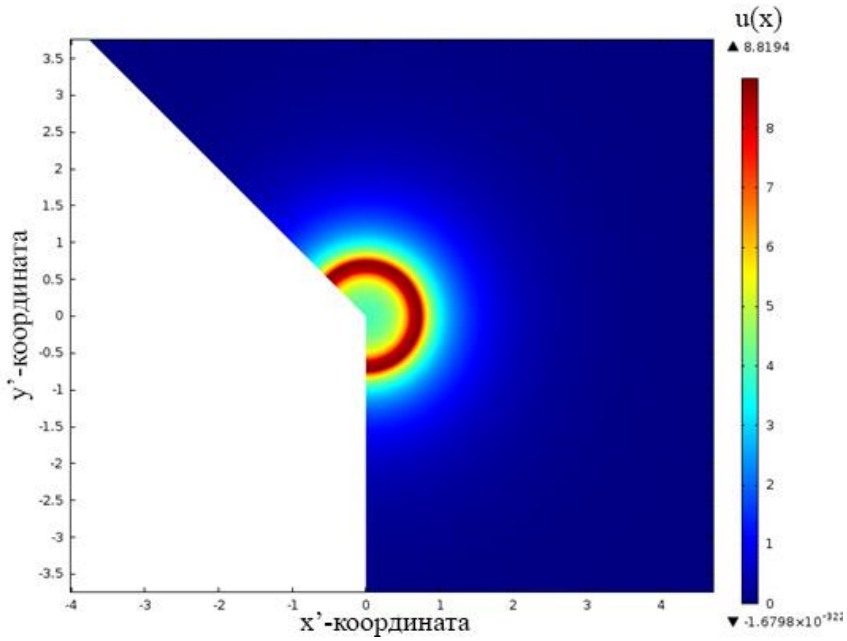


**Рис. 8.** Двумерное распределение электростатического потенциала в пространственных координатах x'–y', полученное с помощью COMSOL MULTIPHYSICS 4.3 для везикулы $C_8$+$C_8$ с параметрами $R = 5l_C$, $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\ l_C$, $C_{salt} = 1$ мМ.

Перейдем к везикулам, образованным в смесях реальных ПАВ, которые различаются длиной алкильных хвостов и площадью поперечного сечения головы. На рис. 9 сопоставляются профили электрического потенциала в везикуле, образованной $C_{16}$TAB ($a_p = 54$ Å$^2$) и SOS ($a_p = 17$ Å$^2$), и в модельной везикуле $C_8$+$C_8$, рассмотренной ранее.

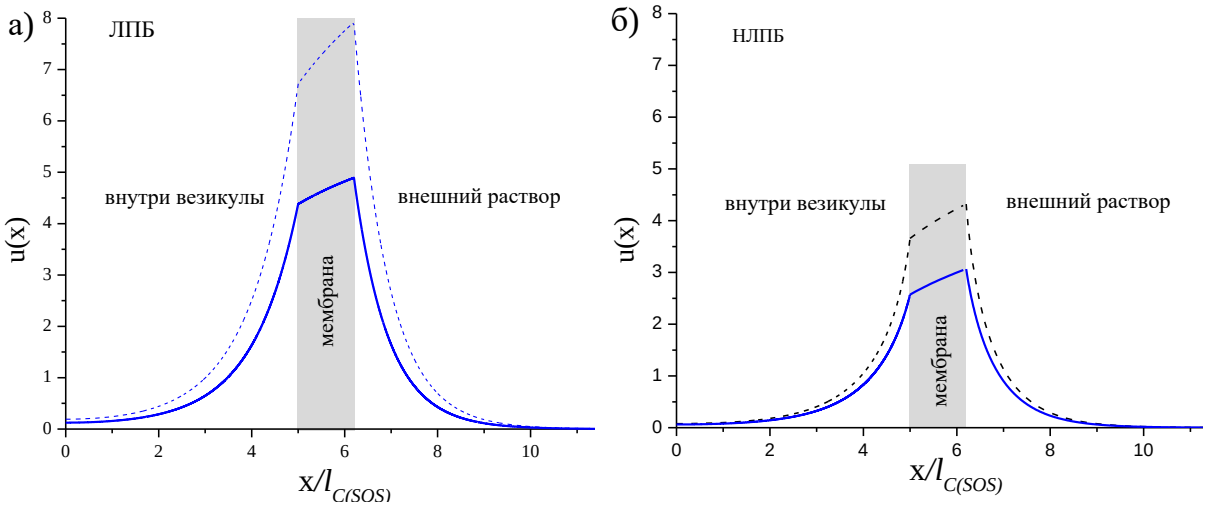


**Рис. 9.** Профиль электростатического потенциала, рассчитанного по теории ЛПБ (а) и по теории НЛПБ (б), для везикулы $C_{16}$TAB+SOS (сплошные линии) и модельной везикулы $C_8$+$C_8$ (пунктирные линии). Параметры системы: $R = 5l_C$, $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\ l_C$, $C_{salt} = 100$ мМ.

Более длинный хвост $C_{16}$TAB сильнее сжат по сравнению с хвостами из 8 углеродов, поэтому средние площади поверхности мембраны на молекулу ПАВ увеличиваются в обоих слоях везикулы. Это приводит к уменьшению поверхностных потенциалов $u_{in}$ и $u_{out}$ для везикулы $C_{16}$TAB+SOS. Как показано на рис. 9, теория ЛПБ дает существенно большие электростатические потенциалы, чем теория НЛПБ. При этом наблюдается частичная компенсация ошибок при расчете разности потенциалов между одинаково заряженными поверхностями мембраны. Благодаря этому, трансмембранные потенциалы теории ЛПБ и теории НЛПБ неплохо согласуются друг с другом.

Для растворов, содержащих 2.0 масс.% общего количества ПАВ (в массовом соотношении 3/7 $C_{16}$TAB/SOS) и разное количество соли NaBr, известны экспериментальные значения дзета-потенциала вблизи поверхности везикул [17]. Эти данные сопоставляются с рассчитанными в настоящей работе значениями электростатического потенциала внешней поверхности везикул с средней мольной долей $C_{16}$TAB в агрегате $\alpha = \alpha_{in}\chi_{in} + \alpha_{out}\chi_{out}$ 0.44 и значениями потенциала на расстоянии длины Дебая от поверхности при разной солености (рис. 10). Для каждого значения $C_{salt}$ были найдены оптимальные везикулы с минимальной свободной

энергией агрегации, электростатический потенциал рассчитывался согласно (ур. 24) теории ЛПБ.

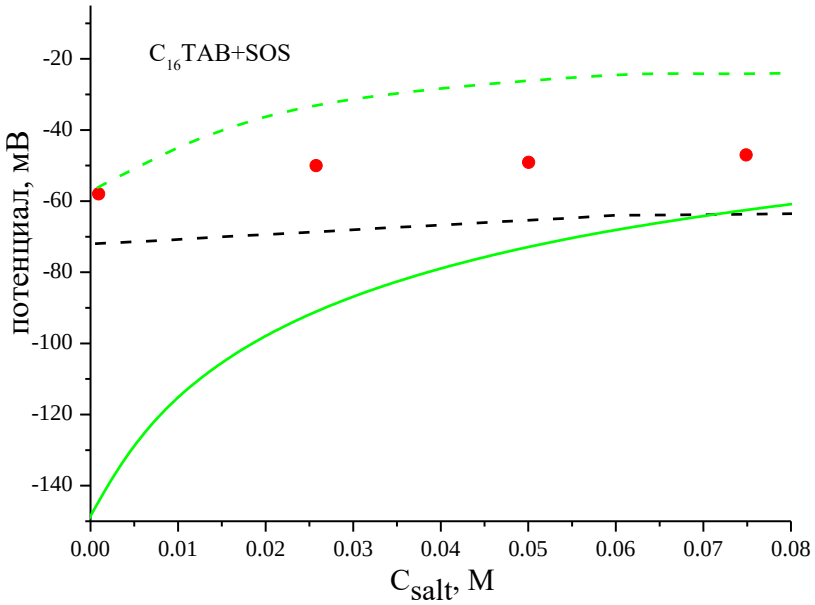


**Рис. 10.** Зависимость электростатического потенциала от солености раствора по теории ЛПБ на внешней поверхности оптимальной везикулы $C_{16}$TAB+SOS (зеленая сплошная линия) и на расстоянии длины Дебая от нее (зеленая пунктирная линия). Символами обозначены экспериментальные данные для дзета-потенциала [17], черной пунктирной линией – расчетные значения поверхностных потенциалов, полученные Бланкштайном по теории НЛПБ для мольной доли $C_{16}$TAB в агрегате 0.44 [10].

Расчитанные значения поверхностного потенциала и потенциала на расстоянии длины Дебая от поверхности позволяют получить оценочный интервал для дзета-потенциала, поскольку граница скольжения находится в пределах диффузионного слоя относительно заряженной поверхности. Как видно из рис. 10, экспериментальные значения дзета-потенциала [17] укладываются в изображенный на рисунке интервал потенциалов. Рассчитанный электростатический потенциал на внешней поверхности везикулы $C_{16}$TAB+SOS при высокой солености разумно согласуется с данными Бланкштайна [10], полученными по теории НЛПБ (рис. 10).

### 4.3. Свободная энергия агрегации и распределение везикул по размерам

Рассмотрим влияние радиуса, толщин и составов везикулы на свободную энергию агрегации $g_{ves}$, рассчитанную по модели Нагаражана-Рукенштейна (раздел 2.2) с учетом полученной аналитически электростатической составляющей $g_{el}$ (ур. 30).

На рис. 11 представлены зависимости свободных энергий $g_{ves}$ и $g_{el}$ от радиуса везикулы $C_{16}$TAB+SOS при постоянных значениях толщин и составов. Кривая имеет минимум при радиусе $R_{\min} \approx 14.5 l_{C(SOS)} \approx 170$ Å, что указывает на минимальный размер везикулы вблизи этого значения. Минимум не является ярко выраженным, что позволяет предполагать полидисперсное распределение по размерам с вероятным появлением более крупных везикул.

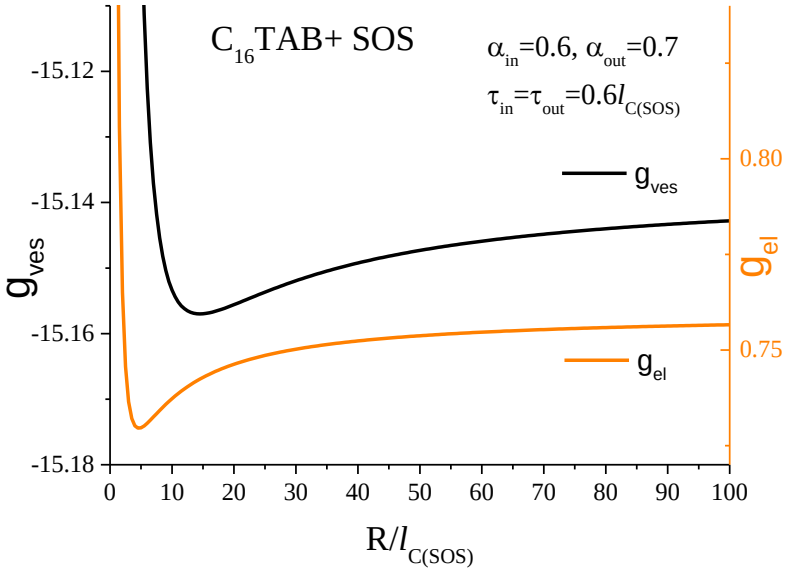


**Рис. 11.** Свободная энергия агрегации везикулы $C_{16}$TAB+SOS (черный цвет) и электростатический вклад в свободную энергию (желтый цвет) в зависимости от радиуса везикулы. Параметры системы: $\alpha_{in} = 0.6$, $\alpha_{out} = 0.7$, $\tau_{in} = \tau_{out} = 0.6\ l_{C(SOS)}$, $C_{salt} = 100$ мМ.

Электростатический вклад в свободную энергию имеет минимум, близкий к экстремуму для $g_{ves}$, но смещенный в область меньших радиусов. Поскольку внешний слой дает больший вклад в свободную энергию, электростатические

взаимодействия снижаются при увеличении кривизны и, соответственно, площади на молекулу внешнего слоя, что способствует уменьшению $u_{out}$ и $g_{el}$.

Зависимости свободной энергии от состава везикулы и соотношения толщин показаны на рис. 12. Поскольку в условиях эксперимента контролируется только брутто-состав ПАВ, влияние состава внутреннего слоя на значение $g_{ves}$ (рис. 12а) рассмотрено при постоянстве средней мольной доли $C_{16}$TAB в агрегате $\alpha$. На рис. 12б одновременно варьируются толщины обоих слоев везикулы при сохранении общей толщины мембраны.

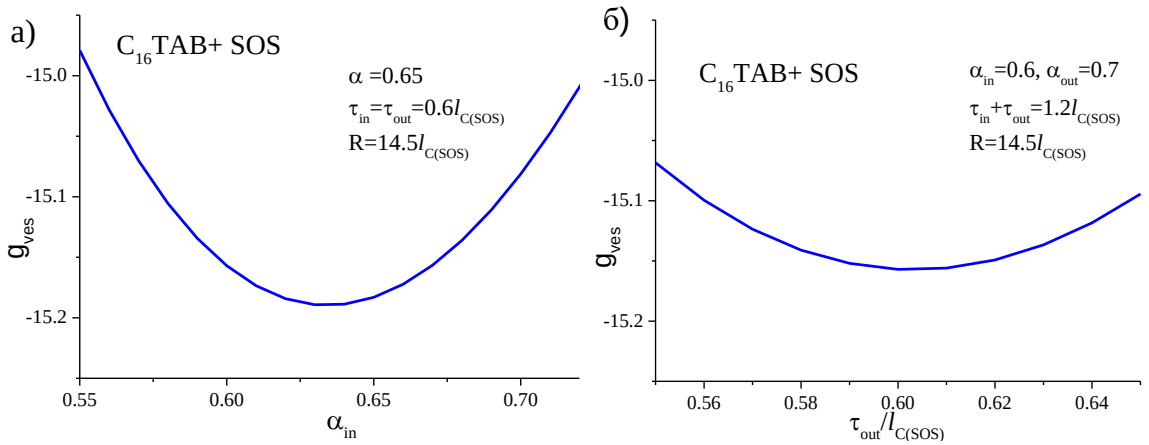


**Рис. 12.** Свободная энергия агрегации везикулы $C_{16}$TAB+SOS в зависимости от состава внутреннего слоя везикулы при постоянстве среднего состава $\alpha = 0.65$ (а) и от толщины внешнего слоя мембраны при постоянстве толщины мембраны $\tau_{in} + \tau_{out} = 1.2\ l_{C(SOS)}$ (б). Остальные параметры системы: $R = 14.5 l_{C(SOS)}$, $C_{salt} = 100$ мМ.

Как видно из полученных графиков, составы слоев оказывают большее влияние на $g_{ves}$, чем толщины. Для везикул выгоднее иметь менее заряженный внутренний слой, так как в нем головы ПАВ находятся ближе друг к другу и сильнее отталкиваются в случае большого количества одноименных зарядов.

Построим распределение по размерам везикул $C_{16}$TAB+SOS (рис. 13) с общей брутто-концентрацией ПАВ 40 мМ и с солевым фоном 100 мМ по алгоритму, представленному в разделе 3.2. Данные представлены для мольных долей катионного ПАВ в системе $y = \frac{X_{C16TAB\ tot}}{X_{C16TAB\ tot} + X_{SOS\ tot}}$ 0.46 и 0.24, которые были получены для

выбранных мономерных долей $C_{16}$TAB $10^{-4}$ и $10^{-6}$. Результаты сопоставим с имеющимися экспериментальными данными о размерах везикул для $y$ =0.1 [4].

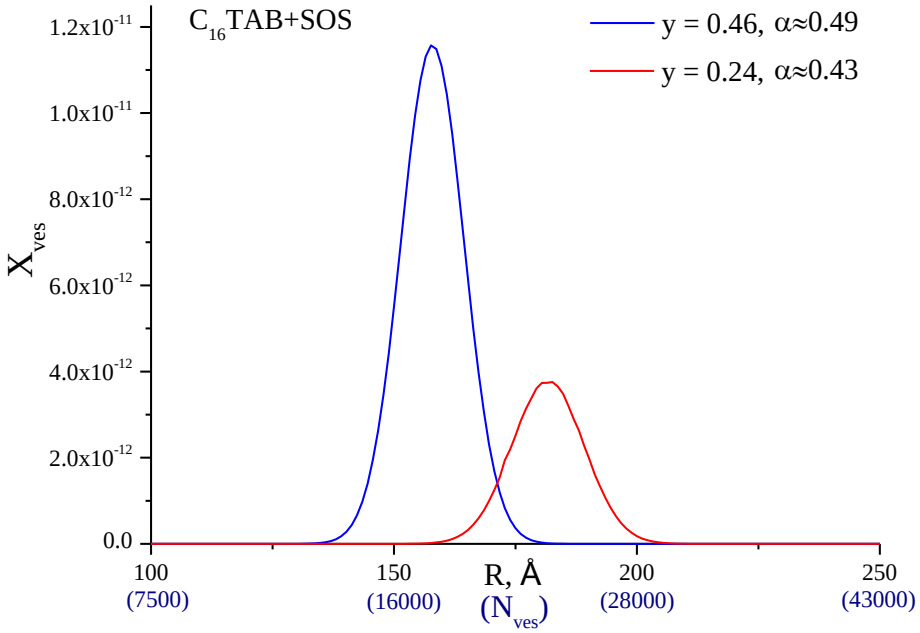


**Рис. 13.** Концентрация везикул в зависимости от их радиуса (чисел агрегации $N_{ves}$) в системе $C_{16}$TAB+SOS с общей брутто-концентрацией ПАВ 40 мМ и солевом фоне 100 мМ. Мольная доля $C_{16}$TAB в системе и средняя мольная доля $C_{16}$TAB в везикуле: y = 0.46 и $\alpha \approx 0.49$ (синяя кривая), y = 0.24 и $\alpha \approx 0.43$ (красная кривая). Мономерная доля $C_{16}$TAB в равновесных условиях $\alpha_{C16TAB\,1} = 10^{-4}$ (синяя кривая), $\alpha_{C16TAB\,1} = 10^{-6}$ (красная кривая).

На оптимальный размер везикул большое влияние оказывает асимметрия гидрофобных хвостов ПАВ. Для небольших везикул вклад внешнего слоя в общую свободную энергию намного больше, чем вклад внутреннего. Для большого (бесконечного) радиуса вклад обоих слоев в общую свободную энергию имеет одинаковое значение. Таким образом, гидрофобная свободная энергия $g_{tr}$ для везикулы минимальна, когда ведущий вклад поступает от внешнего слоя, способствуя стабилизации мелких везикул. В катионной везикуле внешний слой содержит больше $C_{16}$TAB, чем внутренний, и, следовательно, имеет более низкую свободную гидрофобную энергию на молекулу. Ситуация противоположна для анионных везикул $C_{16}$TAB+SOS, где содержание $C_{16}$TAB выше во внутреннем слое, чем во внешнем, и, следовательно, гидрофобный вклад минимален для плоского бислоя.

Поэтому оптимальные анионные везикулы $C_{16}$TAB+SOS имеют больший размер, чем аналогично заряженные катионные.

Эти выводы подтверждает распределение на рис. 13. При переходе к везикулам, содержащим большее количество анионных ПАВ SOS, размер везикул смещается в область больших радиусов: от 160 Å (при $\alpha \approx 0.49$, y = 0.46) до 180 Å (при $\alpha \approx 0.43$, y = 0.24). В эксперименте [4] для таких же значений солевого фона и общей брутто-концентрации ПАВ наблюдают везикулы с радиусом примерно 300 Å для y = 0.1, что качественно согласуется с выводом об увеличении размера везикул при уменьшении доли $C_{16}$TAB в системе.

Рассмотрим аналогичные зависимости (рис. 14) для систем DTAB+SDS ($a_{pDTAB} = 54$ Å$^2$, $a_{pSDS} = 17$ Å$^2$) с общей брутто-концентрацией ПАВ 122 мМ, солевом фоне 100 мМ. Полученные данные сравним с экспериментальными [16].

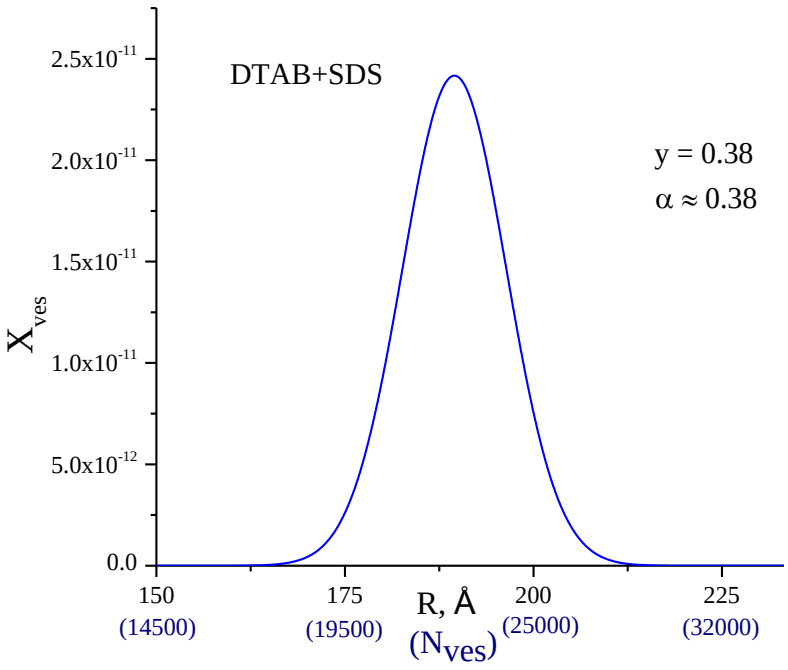


**Рис. 14.** Концентрация везикул в зависимости от их радиуса (чисел агрегации $N_{ves}$) в системе DTAB+SDS с общей брутто-концентрацией ПАВ 122 мМ, солевом фоне 100 мМ. Средняя мольная доля DTAB в везикуле $\alpha \approx 0.38$, мольная доля DTAB в системе y = 0.38. Мономерная доля DTAB в равновесных условиях $\alpha_{DTAB\,1} = 10^{-3}$.

Согласно рис. 14 в системе DTAB+SDS с y = 0.38 характерно образование везикул с диаметром примерно 410 Å с учетом толщины мембраны. Полученный

результат согласуются с данными [16], где в аналогичной системе с y = 0.38 наблюдаются везикулы с диаметром 430-450 Å.

Расчет для катанионной системы, содержащей мицеллообразующую ионную жидкость и классическое анионное ПАВ ($C_{16}$mimCl+SDBS; $a_{pCmim} = 46$ Å$^2$, $a_{pSDBS} = 80$ Å$^2$), предсказывает формирование везикул. Так, для раствора с общей брутто-концентрацией ПАВ 150 мМ и 30 мМ добавленной соли модель дает везикулы среднего диаметра 271 Å и 308 Å с достаточно узким распределением по размерам при мольных долях $C_{16}$mimCl в системе y = 0.56 и y = 0.66, соответственно. Эксперимент [19] (SI) для бессолевого раствора $C_{16}$mimCl+SDBS при общей брутто-концентрации ПАВ 150 мМ также свидетельствует о переходах мицеллы-везикулы-мицеллы при последовательном изменении соотношения ПАВ. Однако для сходных значений y обнаруживается существенно полидисперсное распределение везикул диаметрами от 100 до 10000 Å с пиком около 1000 Å.

**Выводы**

В рамках данной работы были получены аналитические выражения для трансмембранного потенциала (ур. 26) и электростатического вклада в свободную энергию (ур. 30) с учетом эффекта регулирования зарядов внутренней и внешней частей везикулы в растворе соли.

Изучены зависимости трансмембранного потенциала и свободной энергии агрегации катанионных везикул от геометрических параметров везикулы как для модельных агрегатов, так и для реальных систем. Выявлено нетривиальное поведение трансмембранного потенциала при изменении солевого фона: появление экстремума в случае, когда внешний слой мембраны заряжен сильнее, чем внутренний. Предложены простые формулы (ур. 33, 34), позволяющие определить условия образования экстремума. Дано физическое объяснение появления экстремума трансмембранного потенциала.

Получено выражение (ур. 35) для определения pH во внутреннем растворе везикулы и представлена зависимость этой величины от кислотности окружающего раствора. Продемонстрирована ситуация, когда слабощелочной раствор находится внутри равновесной везикулы, окруженной слабокислым раствором.

Разработан вычислительный алгоритм и программное обеспечение на языке FORTRAN, позволяющие решать уравнение материального баланса для известной суммарной брутто-концентрации двух ПАВ с учетом агрегативного равновесия в растворе, включающем монодисперсные сферические мицеллы, полидисперсные стержнеобразные мицелл и полидисперсные везикулы. Получены данные распределения везикул по размерам для систем $C_{16}$TAB+SOS, DTAB+SDS, $C_{16}$mimCl+SDBS, выявлены закономерности, влияющие на размеры агрегатов, проведено сопоставление результатов с экспериментальными данными. Обсуждены пределы применимости модели. Показано, что при среднем и умеренно высоком солевом фоне предлагаемая модель верно передает наблюдаемые тенденции агрегативного поведения растворов катанионных ПАВ.

**Благодарности**

Работа выполнялась при финансовой поддержке РФФИ (грант 18-03-00698а) и РНФ (грант 20-13-00038).

**Список цитированной литературы**


1. Du J., O'Reilly R.K. Advances and challenges in smart and functional polymer vesicles // Soft Matter. 2009. Vol. 5, № 19. P. 3544–3561.
2. Scermino L. et al. pH-responsive micellization of an amine oxide surfactant with branched hydrophobic tail // J. Mol. Liq. Elsevier B.V., 2020. Vol. 316. P. 113799.
3. Almgren M. Mixed micelles and other structures in the solubilization of bilayer lipid membranes by surfactants // Biochim. Biophys. Acta - Biomembr. 2000. Vol. 1508, № 1–2. P. 146–163.
4. Bergström L.M. et al. Spontaneous transformations between surfactant bilayers of different topologies observed in mixtures of sodium octyl sulfate and hexadecyltrimethylammonium bromide // Langmuir. 2014. Vol. 30, № 14. P. 3928–3938.
5. Araste F. et al. Self-assembled polymeric vesicles: Focus on polymersomes in cancer treatment // J. Control. Release. Elsevier B.V, 2021. Vol. 330. P. 502–528.
6. Cho J.A. et al. Exosomes: A new delivery system for tumor antigens in cancer immunotherapy // Int. J. Cancer. 2005. Vol. 114, № 4. P. 613–622.
7. Wen, Z., You, X., Liu, B., Zheng, Z., Pu, Y., Jiang, L., & Li Q. Formation of atractylone liposomes by rapid expansionfrom supercritical to surfactant solution // Asia-Pac. J. Chem. Eng. 2011. Vol. 6, № 4. P. 624–630.
8. Israelachvili J.N., Mitchell D.J., Ninham B.W. Theory of self-assembly of lipid bilayers and vesicles // BBA - Biomembr. 1977. Vol. 470, № 2. P. 185–201.
9. Nagarajan R., Ruckenstein E. Self-Assembled systems // Experimental Thermodynamics. Equations / ed. Sengers, J.V.; Kayser, R.F.; Peters, C.J.; White H.J. Science, Amsterdam: Elsevier, 2000. Vol. 5, № C. 589-749 p.
10. Yuet P.K., Blankschtein D. Molecular-thermodynamic modeling of mixed cationic/anionic vesicles // Langmuir. 1996. Vol. 12, № 16. P. 3802–3818.
11. Mille M., Vanderkooi G. Electrochemical properties of spherical polyelectrolytes: II.

Hollow sphere model for membranous vesicles // J. Colloid Interface Sci. Academic Press, Inc., 1977. Vol. 61, № 3. P. 455–474.

12. Israelachvili J.N. Intermolecular and Surface Forces // Surface Science Reports. 3rd ed. Burlington, MA: Academic Press, 2011. Vol. 556 p.

13. Danino D., Bernheim-Groswasser A., Talmon Y. Digital cryogenic transmission electron microscopy: An advanced tool for direct imaging of complex fluids // Colloids Surfaces A Physicochem. Eng. Asp. 2001. Vol. 183–185. P. 113–122.

14. Bergström M., Pedersen J.S. A Small-Angle Neutron Scattering (SANS) Study of Tablet-Shaped and Ribbonlike Micelles Formed from Mixtures of an Anionic and a Cationic Surfactant // J. Phys. Chem. B. 1999. Vol. 103, № 40. P. 8502–8513.

15. Ghazal A. et al. Microfluidic Platform for the Continuous Production and Characterization of Multilamellar Vesicles: A Synchrotron Small-Angle X-ray Scattering (SAXS) Study // J. Phys. Chem. Lett. 2017. Vol. 8, № 1. P. 73–79.

16. Kakehashi R., Karlsson G., Almgren M. Stomatosomes, blastula vesicles and bilayer disks: Morphological richness of structures formed in dilute aqueous mixtures of a cationic and an anionic surfactant // J. Colloid Interface Sci. Elsevier Inc., 2009. Vol. 331, № 2. P. 484–493.

17. Brasher L.L., Herrington K.L., Kaler E.W. Electrostatic Effects on the Phase Behavior of Aqueous Cetyltrimethylammonium Bromide and Sodium Octyl Sulfate Mixtures with Added Sodium Bromide // Langmuir. 1995. Vol. 11, № 11. P. 4267–4277.

18. Kaler E.W. et al. Phase behavior and structures of mixtures of anionic and cationic surfactants // J. Phys. Chem. 1992. Vol. 96, № 16. P. 6698–6707.

19. Dutta R. et al. Micelle-vesicle-micelle transition in aqueous solution of anionic surfactant and cationic imidazolium surfactants : Alteration of the location of different fluorophores // J. Colloid Interface Sci. Elsevier Inc., 2017. Vol. 490. P. 762–773.

20. Regev O. et al. Vesicle Formation and General Phase Behavior in the Catanionic Mixture SDS−DDAB−Water. The Cationic-Rich Side // J. Phys. Chem. B. 2002. Vol.

103, № 39. P. 8353–8363.

21. Marques E.F. et al. Vesicle Formation and General Phase Behavior in the Catanionic Mixture SDS-DDAB-Water. The Cationic-Rich Side // J. Phys. Chem. B. 1999. Vol. 103, № 39. P. 8353–8363.

22. Русанов А.. Мицеллообразование в растворах поверхностно-активных веществ // СПб. Наука. 1992. P. 280 с.

23. Israelachvili J.N., Mitchell D.J., Ninham B.W. Theory of self-assembly of hydrocarbon amphiphiles into micelles and bilayers // J. Chem. Soc. Faraday Trans. 2 Mol. Chem. Phys. 1976. Vol. 72. P. 1525–1568.

24. Likhtman A.E., Semenov A.N. Stability of the OBDD Structure for Diblock Copolymer Melts in the Strong Segregation Limit // Macromolecules. 1994. Vol. 27, № 11. P. 3103–3106.

25. Hildenrand F.B. Advanced calculus for applications. 2–nd ed. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1976. 733 p.

26. Winterhalter M., Helfrich W. Effect of surface charge on the curvature elasticity of membranes // J. Phys. Chem. 1988. Vol. 92, № 24. P. 6865–6867.

27. Yuet P.K., Blankschtein D. Approximate Expressions for the Surface Potentials of Charged Vesicles // Langmuir. 1995. Vol. 11, № 6. P. 1925–1933.

28. Emelyanova, Ksenia A.; Sorina, Polina O.; Victorov A.I. Transmembrane potential in vesicles formed by catanionic surfactant mixtures in an aqueous salt solution // Phys. Chem. Chem. Phys. Royal Society of Chemistry, 2020. Vol. 22, № 45. P. 26438–26451.

29. Андреев В.А. Моделирование образования, роста и ветвления мицеллярных агрегатов в растворах ионных поверхностно-активных веществ: дисс. к.х.н. // Санкт-Петербургский гос. университет, СПб. 2006.

## Приложения

### А. Программное обеспечение FORTRAN для расчета агрегативных характеристик водно-солевых растворов смеси двух ПАВ с учетом образования сферических и стержнеобразных мицелл и везикул

Во входном файле «infile» указываются исходные параметры системы: название и вид двух ПАВ, их молекулярные параметры, брутто-концентрации, концентрация 1:1 электролита, температура. Пример файла представлен ниже:

GIVE SURFACTANT1 AND SURFACTANT2 NAMES (up to 20 characters each):
CTAB
SOS
SPECIFY SURFACTANT1 AND SURFACTANT2 TYPES ( cationic, anionic, or nonionic. For ionic surfactants NEGATIVE DIPOLE LENGTH MUST be given):
cationic
anionic
TEMPERATURE (IN K):
298.16
CHARACTERISTICS OF SURFACTANT1:
NUMBER OF CARBONS IN THE TAIL = 16.0
POLAR HEAD SCREENING AREA (IN SQ.ANGSTROMS)= 54.0
CLOSEST APPROACH OF COUNTERION TO MICELLE (IN ANGSTROMS)= 3.45
SOLUBILITY PARAMETER = 16.76
DIPOLE LENGTH (positive value, IN ANGSTROMS, only for zwitterionic surfactant)= -1.0
CHARACTERISTICS OF SURFACTANT2:
NUMBER OF CARBONS IN THE TAIL = 8.0
POLAR HEAD SCREENING AREA (IN SQ.ANGSTROMS)= 17.0
CLOSEST APPROACH OF COUNTERION TO MICELLE (IN ANGSTROMS)= 5.45
SOLUBILITY PARAMETER = 16.76
DIPOLE LENGTH (positive value, IN ANGSTROMS, only for zwitterionic surfactant)= -1.0
GROSS CONCENTRATIONS IN SOLUTION (in M):
0.20E-01 ! SURFACTANT1
0.20E-01 ! SURFACTANT2
0.10E-00 ! SALT

Основные выводные файлы следующие. «Aggreg_details» указывает средние составы, числа агрегации и размер агрегатов каждого типа. «CMCmix» указывает для каждой мономерной доли ПАВ $\alpha_{A1}$ полученные значения концентраций $X_{1A}$, $X_{1B}$, $X_{totA}$, $X_{totB}$. Файлы «COREsph», «CORECyl», «COREves» предоставляют информацию о вкладах в свободную энергию агрегации для агрегатов с максимальной концентрацией: сферических, стержнеобразных мицелл и везикул. «Distrib_sphANDcyl» демонстрирует распределение по размерам сферических и стержнеобразных мицелл, «Distrib_ves» - везикул.

Код программного обеспечения представлен ниже.

```
implicit REAL*8 (A-H,O-Z)                              !  MAIN   PROGRAM
integer, parameter :: ch = 9

real*8 MconstA(ch), MconstB(ch)
!  1     2    3    4    5    6    7     8      9
!  Nc    Vs   ls   a0   ap   delta deltaH  logic   ddip
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
    nzero,krDeb,dchpotCH2,dchpotCH3,gXgsph,g2Xgsph,&
          minx,miny,micellesgXg,Xgsph,XAsph,Cadd,CsurA,CsurB
 CHARACTER*20 EX1,EX2
 CHARACTER*8 MDL
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
 COMMON /Y/ H,VheadA,VheadB,VtailA,VtailB,CH2lgth
 COMMON /C/ QT,RT,T1,TT,R(10),RS(10,10),Q(10),QS(10,10)
 COMMON /word/ MDL
real*8 volume,length, MA,MB,S0,S1,S2,XArod, nsph,lsinf,lssup,Y,XtotA,XtotB
real*8 njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,NagA_pl,NagB_pl,NagA_tor,NagB_tor
real*8 alfavesopt,alfaiopt,alfaoopt,Rvesopt,gvesopt,tauiopt,tauoopt
real*8 nves,gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
logical Ybig1
REAL*8, PARAMETER :: lqvad = 4.6*4.6
real*8 X1A,X1B,alfaA1,cylopt(1,2),sphopt(1,2),capopt(1,2),alfaBmic,KoefRasp
common /monomers/ X1A,X1B,alfaA1
common /lsformin/ lsinf,lssup
common /exitvalue/ S0,S1,S2,XArod, nsph,gXgsph,g2Xgsph,Xgsph,XAsph,XBsph,XBrod
common/exitjun/ njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun
common/exitves/gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
common /geomopt/ cylopt,capopt,sphopt,Y
common /optves/alfavesopt,alfaiopt,alfaoopt,Rvesopt,gvesopt,tauiopt,tauoopt,nves
common /optjun/ alfa_jun, bjun,cjun,g_j
COMMON/optpl/r_patch,alfa_pl,g_p
common /optcyl/ alfacylopt,Rcylopt,gcylopt
common /optsph/ alfasphopt,Rsphopt,gsphopt
common/gross_mfrac/XtotA,XtotB
common /out/ ldamp
! Cadd = 0.0d0    ! added salt  CsurA,  CsurB   come through infile
T =        273.16d0+ 25.0d0 !30.0d0
T1 = T
Pi = 3.1415926
psi = .55
k = 1.38d-16
```

```
e = 4.80286d-10
Nav = 6.022142d+23

OPEN(UNIT=11,FILE='COREsph.txt')
OPEN(UNIT=12,FILE='SHELLcyl.txt')
OPEN(UNIT=15,FILE='SHELLsph.txt')
OPEN(UNIT=14,FILE='COREcyl.txt')
OPEN(UNIT=16,FILE='COREjun.txt')
OPEN(UNIT=17,FILE='nonidINshell.txt')
OPEN(UNIT=18,FILE='COREves.txt')
OPEN(UNIT=30,FILE='result_new.txt')

OPEN (3,file='Aggreg_details.txt')
Open (4, file='Free energies.txt')
OPEN (2,file='CMCmix.txt')
OPEN (21,file='MicCompCMC.txt')

write (2,*) '  T      = ', T
write (3,*) '  T      = ', T
write (4,*) '  T      = ', T
call datain (ex1,ex2)
sigmaw = 72. - .16 * (T - 298.)
  MA = 14.02709*(MconstA(1) - 1.) + 15.03506
  MB = 14.02709*(MconstB(1) - 1.) + 15.03506
sigmaA = 35. - 325. * MA ** (-2. / 3.) - .098 * (T - 298.)
sigmaB = 35. - 325. * MB ** (-2. / 3.) - .098 * (T - 298.)
sigmasA = sigmaA + sigmaw - 2. * psi * dsqrt(sigmaA * sigmaw)
sigmasB = sigmaB + sigmaw - 2. * psi * dsqrt(sigmaB * sigmaw)
dchpotCH2 = 5.85 * dlOG(T) + 896. / T - 36.15 - .0056 * T
dchpotCH3 = 3.38 * dLOG(T) + 4064. / T - 44.13 + .02595 * T
epsilon = 87.74 * dEXP(-.0046 * (T - 273.))
XtotA = CsurA/(CsurA+CsurB+998.2d0/18.0152d0)                    !                    may be better  55,(5)
XtotB = CsurB/(CsurA+CsurB+998.2d0/18.0152d0)                    !

OPEN(UNIT=88,FILE='model.txt')
Read (88,*) Mdl
if(Mdl.eq.'new') then
call indsurf (H,VheadA,VheadB,VtailA,VtailB,CH2lgth,ntailA,ntailB)
MconstA(2)=VtailA
MconstB(2)=VtailB
Atail = ntailA
Btail = ntailB
MconstA(3) = length (Atail)
MconstB(3) = length (Btail)
```

```
MconstA(1)=Atail
MconstB(1)=Btail
else
  MconstA(2) = volume (MconstA(1), T)
  MconstA(3) = length (MconstA(1))
  MconstB(2) = volume (MconstB(1), T)
  MconstB(3) = length (MconstB(1))
end if

If (MconstA(5)>lqvad) then
  MconstA(4) = lqvad
else
  MconstA(4) = MconstA(5)
end if
If (MconstB(5)>lqvad) then
  MconstB(4) = lqvad
else
  MconstB(4) = MconstB(5)
end if
!MconstB(4) = lqvad       !MconstB(5)
!MconstA(4) = lqvad       !MconstA(5)
write (2,*) ''
write (2,*)  'ComponentA=',EX1,'ComponentB=',EX2
write (3,*)  'ComponentA=',EX1,'ComponentB=',EX2
write (4,*)  'ComponentA=',EX1,'ComponentB=',EX2
write (2,*)  ' '
write (2,1002)
1002  format(22X,'Nc                Vs',13X,'ls',12X,'a0',12X,'ap',8X,'Ion  apprch',4X,'Solub  Par',2X,'1/2/3(+/-/nonionic)',2X,'D_dip')
write (2,'(A12,9(4X,F10.4))') '  MconstA : ', MconstA
write (2,'(A12,9(4X,F10.4))') '  MconstB : ', MconstB
lsinf = 0.3*min(MconstA(3),MconstB(3))                                    !  try  to  change  worked with 0.3
lssup = max(MconstA(3),MconstB(3))

write  (2,'(5(A10,4X,E12.5,2X))')  'lsinf=',lsinf,'lssup=',lssup,'Csalt  [M]=',  Cadd,'CsurA  [M]=',CsurA,'CsurB [M]=', CsurB
! ************************************************************************
  write (2,*) ''
if(CsurA==0.and.CsurB==0.)then                    !  CMC  calculation
  write (2,1001)
else                                                                      !  material balance calculation
  write (2,1021)
endif
  write (11,1011)
```

```
      write (12,1012)
      write (15,1012)
      write (14,1011)
      write (17,1017)
      write (16,1016)
      write (18,1022)

 1017 format (4X,'XREM(1)',4X,'XREM(2)',4X, 'XREM(3)',3X,'gmxbond',6X,'gmxwater', 5X, 'gmxbond+
gmxwater')
 1001 format(4X,'alAMon alAMic alAGross CMC(mol/l) ',5X,'gaverage gsphOPT
',8X,'X1A',10X,'X1B',10X,'tolerance,%')
 1021 format(4X,'alAMon alAMic alAGross A+BinMic(mol/l) ',1X,'gaverage gsphOPT
',8X,'X1A',10X,'X1B',10X,'tolerance,%',2X,' totalA,M',2X,' totalB,M')
 1011 format (5X,'alAMic',6X,'Def',10X, 'Transf',9X,'Mix',10X,'Interf',8X,'Steric', 8X,'Zwitter',8X,
'IonLPB',8X,'NLPB', 8X, 'Total',8X,'R [A]',15X,'H')
 1016 format (2X,'plane/tor',2X,'alAMic',6X,'Def', 10X,'Transf',9X,'Mix',10X,'Interf',8X,
'Steric',8X,'Zwitter',8X,'IonLPB',8X,'NLPB',8X, 'Total',8X,'R [A]',8X,'<n_aggr>')
 1022 format (5X,'alAMic',6X,'Def',10X,'Transf', 9X,'Mix',10X,'Interf',8X,'Steric',8X,'Zwitter',
8X,'IonLPB',8X,'NLPB',8X, 'Total',8X,'R [A]',10X,'R+tau [A]',10X,'alfao',8X,'alfai',15X)
 1012 format (3X,'alAMic',6X,'FiA',6X,'FiB',6X,'water',8X,'Gloc',11X,'GMXbond',6X,'Gsh')
      write (3,*) ''
      write (3,1013)
 1013 format (9X,'alAMon alAMic alA_tot alA_sph',6X,'alA_cyl alA_jun
alA_ves',5X,'Rsph/ls',7X,'Rcyl/ls',7X,'Rves/ls') !'btor/ls', 6X,'ctor/btor',&
 !5X,'nA_av_sph nB_av_sph nA_av_cyl nB_av_cyl nA_av_jun nB_av_jun',5X,'nsph_opt
njun_opt <n>',12X,'nA_pl',8X,'nB_pl',8X,'nA_tor',8X,'nB_tor')
      write (4,*) ''
      write (4,1014)
 1014 format(3X,'alAMon alAMic alA_tot',6X,' g_sph g_cyl g_ves g_jun
g_pl',8X,'tolerance,%')
      i=0
      alfaA1=0.001d0
       do while (alfaA1.LT.1.001d0)
      i=i+1
      ldamp=-2                                                    ! no printing while solving
      XtotA = CsurA/(CsurA+CsurB+998.2d0/18.0152d0)
      XtotB = CsurB/(CsurA+CsurB+998.2d0/18.0152d0)

      if(alfaA1==0)then
      XtotA = 0.0d0
      XtotB = CsurB/(CsurB+998.2d0/18.0152d0)
      endif
      if(dabs(alfaA1-1.0d0)<1.0d-15)then
      XtotA = CsurA/(CsurA+998.2d0/18.0152d0)
      XtotB = 0.0d0
      endif
```

```
!a-starting point, del-starting step, eps  tolerance in x    minx-solution
   call  dihotom (1.6d-3, -1.d-6,1.d-20,minx,miny)              ! works for SDS  at accuracy  d-20, d-15  and d-10
        X1A = minx*alfaA1
!       write(*,*)'X1=',minx                                                                    !(1.6d-5, -1.d-8,1.d-15,minx,miny)  - for the first 1st 2 points goes OK
   X1B = minx*(1.0d0 - alfaA1)
print *, 'Ybig1===============', Ybig1, Y
        call micelles(minx,Ybig1,micellesgXg)

        nsph = 4./3.*3.1415926*Rsphopt**3/vs(alfasphopt)
print *,'micelles== ',minx,micellesgXg,Ybig1
                ldamp = 2    !                  print out at solution from  subroutines
                aa = chpotcyl(cylopt)
     aa=chpot(sphopt)
!print *,'parpl== ',r_patch,alfa_pl
!!              aa=g_pl(r_patch,alfa_pl)
!print *, 'parjun==  ',bjun,cjun,alfa_jun
!!              aa=g_tor(bjun,cjun,alfa_jun)
                NagA_pl = alfa_pl*v_patch(r_patch,cjun)/MconstA(2)
 ! volume occupied by A-tails/A-tail volume          Aggregation number in planar patch
                NagB_pl = (1.0d0-alfa_pl)*v_patch(r_patch,cjun)/MconstB(2)                      !
                NagA_tor = alfa_jun*v_tor_b(bjun,cjun)/MconstA(2)
 ! Aggregation number in toroidal rim: NB: alfa_jun = alfator in this version(CHECK  vs(alfa) in optimization)
                NagB_tor = (1.0d0-alfa_jun)*v_tor_b(bjun,cjun)/MconstB(2)
!               njun=  NagA_pl+NagB_pl + NagA_tor + NagB_tor
                call chpotves
                  !search minimum chpot,write distribution by Nagg
!       calculate  average  aggregation  numbers  and  composition  of micelles**********************************************************************
! nA_av_sph=XAsph/Xgsph          nA_av_cyl = XArod/S0     na_av_jun = XA_jun/Xg_jun   nsph njun average and optimal  aggregation numbers
!       alfaAmic  =  (XAsph+XArod+Xa_jun)/micellesgXg                                   alfaA_tot  =  (X1A+XAsph+XArod+Xa_jun)/(micellesgXg+X1A+X1B)   average composition
! alfa_jun, bjun,cjun,g_j, cylopt, sphopt,
alfaA_tot =  X1A+XAsph+XArod+XA_jun+XA_ves
write(*,*)'!!aAtot=',alfaA_tot,XA_ves
!(X1A+XAsph+XArod+XA_jun)/(micellesgXg+X1A+X1B)   !average composition
alfaB_tot =  X1B+XBsph+XBrod+XB_jun+XB_ves
   write(*,*)'sph,rod,jun,ves',Xgsph,s0,Xg_jun,Xg_ves
!(X1B+XBsph+XBrod+XB_jun)/(micellesgXg+X1A+X1B)
        alfaAmic=(XAsph + XArod+Xa_jun+XA_ves)/(S1 + gXgsph + gXg_jun+gXg_ves)
        alfaBmic=1.0d0-alfaAmic
         write(*,*)'gXgsph,rod,jun,ves',gXgsph,S1,gXg_jun,gXg_ves
        write(*,*)'!!aAmic=',alfaAmic
if(alfaA1/=0.and.alfaA1/=1.) KoefRasp=alfaBmic/(1.0d0-alfaA1)
```

```
                print *,minx, micellesgXg,micellesgXg/0.018,minx/0.025/0.018

            print *,alfaA1,(XAsph + XArod+Xa_jun+XA_ves)/(S1 + gXgsph+ gXg_jun+gXg_ves),&          ! printing solvent-free monomer fraction of comp A  and  fraction of A the in micelle

            (minx*alfaA1 + XAsph + XArod+Xa_jun+XA_ves)/(minx + gXgsph + S1+ gXg_jun+gXg_ves) ,&      ! printing gross (monomers+micelles) fraction of comp A  (solvent-free)

            (S1 + gXgsph+ gXg_jun+gXg_ves)/(S0 + Xgsph+Xa_jun+XA_ves),&

                (S2 + g2Xgsph+g2Xg_jun+g2Xg_ves)/(S1 + gXgsph+ gXg_jun+gXg_ves),&    ! printing number-averaged and weight-averaged aggregation numbers

                cylopt,sphopt                                                    ! printing optimal radius and optimal composition:  for cylinders and then for spheres

        !!      if (Xg_jun.LT.1.0d-300) Xg_jun=1.d-300

                if (S0.LT.1.0d-300) S0=1.d-300

                if (Xgsph.LT.1.0d-300) Xgsph=1.d-300

            write (21,*) alfaA1, alfaAmic,micellesgXg/0.018


        !     print *, 'cylinders: S0 = ','S1 = ',S1, 'spheres: gXgsph = ',gXgsph,'junctions: Xg_jun = ',Xg_jun,'Total aggregated= ',micellesgXg,'XtotA= ',XtotA,'XtotB= ',XtotB,minx

            write (*,'(/,2X,A16,3X,E13.4,3X,A7,E13.4,/,2X,A16,3X,E13.4,3X,A7,E13.4,/2X,A14,5X,E13.4,2X,A5,3X,E13.4,/,2X,A19,E13.4,2X,A8,E13.4,/,2(25X,A16,3X,E13.4/),2X,A8,11X,E13.4,4X,A8,9X,E13.4,/)')&

                'Spheres: Xgsph= ', Xgsph,'gXgsph=',gXgsph,'Vesicles: Xg_ves= ', Xg_ves,'gXg_ves=',gXg_ves,&

                'Cylinders: S0=',S0,' S1= ', S1,'Junctions: Xg_jun= ',Xg_jun,' gXgjun=',(njun*Xg_jun),&

                'Tot.aggregated= ',micellesgXg,'Tot.monomers=  ',minx,'Total A=',XtotA,'Total B=',XtotB

                 write (*,'(2X,A17,E11.4,3(1X,A4,E11.4)//,2X,A12,E10.4,2X,A5,E10.4,2X,A8,F8.3)')&

                 'Free energy: sph=',gsphopt,'cyl=',gcylopt,' pl=',g_p,'ves',gvesopt,'jun=',g_j,'Aggr#: Nsph=',nsph,'Nves=',nves,'Njun=',njun,'c/b jun=',(cjun/bjun)

        !       if (s0==0.0d0)cycle

        !ldamp=-2

        !       if (i==7) call shape_distrib                                     ! output of distributions for a desired composition-point


        rtrtr=1.0d-15

        !stop

                write (3,'(7(3X,E10.5),18(1X,E13.5),L2)') alfaA1,alfaAmic,alfaA_tot, sphopt(1,2), cylopt(1,2), alfa_jun,alfavesopt, sphopt(1,1)/lssup, cylopt(1,1)/lssup,rvesopt/lssup !!bjun/lssup , cjun/bjun,&

        !!      XAsph/Xgsph,XBsph/Xgsph,XArod/S0,XBrod/S0,XA_jun/Xg_jun,XB_jun/Xg_jun,nsph,njun,(S1 + gXgsph+gXg_jun)/(S0 + Xgsph+Xg_jun),NagA_pl,NagB_pl,NagA_tor,NagB_tor,Ybig1


        if(CsurA==0.and.CsurB==0.)then                                   !  CMC  calculation

                write                                                    (4,'(3(2X,E10.5),6(1X,E13.5),L2)') alfaA1,alfaAmic,alfaA_tot,gsphopt,gcylopt,gvesopt,g_j,g_p,100.0d0*(micellesgXg/minx - 1.0d0)

                write (2,'(3(2X,E10.5),6(2X,E13.5))') alfaA1,alfaAmic,(alfaA1+alfaAmic)/2.0,micellesgXg/0.018,(S1 + gXgsph+gXg_ves)/(S0 + Xgsph+Xg_ves),nsph,X1A,X1B,&

                100.0d0*(micellesgXg/minx - 1.0d0)          !(100*sqrt(miny*(X1A+X1B)**3/micellesgXg**2))

        else                                                             !  material balance calculation
```

```
                write                                                    (4,'(3(2X,E10.5),6(1X,E13.5),L2)')
alfaA1,alfaAmic,alfaA_tot,gsphopt,gcylopt,gvesopt,g_j,g_p,micellesgXg+minx                    !100.0d0*(1.0d0-
(micellesgXg+minx)/(XtotA+XtotB) )


          if(alfaA1<1.0d-15.or.dabs(alfaA1-1.0d0)<1.0d-15)then
           write        (2,'(2(2X,E10.5),7(2X,E13.5))')        alfaA1,alfaAmic,alfaA_tot,micellesgXg/0.018,(S1        +
gXgsph+gXg_ves)/(S0 + Xgsph+Xg_jun+Xg_ves),nsph,X1A,X1B,&              !          (alfaA1+alfaAmic)/2.0
                100.0d0*(1.0d0-(micellesgXg+minx)/(XtotA+XtotB))
          else
           write        (2,'(2(2X,E10.5),9(2X,E13.5))')        alfaA1,alfaAmic,alfaA_tot,micellesgXg/0.018,(S1        +
gXgsph+gXg_ves)/(S0 + Xgsph + Xg_jun + Xg_ves),nsph,X1A,X1B,&          100.0d0*(1.0d0-
(micellesgXg+minx)/(XtotA+XtotB)                    ),alfaA_tot/0.018,alfaB_tot/0.018                !100.0d0*(1.0d0-
(micellesgXg+minx)/(XtotA+XtotB) ),100.0d0*(1.0d0-alfaA_tot/XtotA),100.0d0*(1.0d0-alfaB_tot/XtotB)
           write (*,'(2X,A20,E11.4)') 'tot.surf.B/A ratio= ',     alfaB_tot/alfaA_tot
          endif
         endif
          alfaA1=alfaA1+0.1d0
          end do                !   alfaA1  cycle
        close(2)
        close(3)
        close(4)
        close(11)
        close(12)
        close(14)
        close(15)
        close(16)
        close(17)
        close(88)
        close(21)
        end


        !**********************************************************************
        subroutine datain (ex1,ex2)
        implicit REAL*8 (A-H,O-Z)
        integer, parameter :: ch = 9
        real*8 MconstA(ch), MconstB(ch)
        !   1     2     3     4     5     6     7      8       9
        !  Nc     Vs    ls    a0    ap   delta deltaH  logic   ddip
        REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
            nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
         CHARACTER*20 EX1,EX2
         CHARACTER*8 TY(2)
         COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
              nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
```

```
OPEN(UNIT=13,FILE='infile.txt')
     READ (13,507)
 507  FORMAT (15X)
     Read (13,*) Ex1,Ex2
     READ (13,507)
     Read (13,*) Ty
if(Ty(1).eq.'cationic')MconstA(ch-1)=1.0
if(Ty(1).eq.'anionic ')MconstA(ch-1)=2.0
if(Ty(1).eq.'nonionic')MconstA(ch-1)=3.0
if(Ty(2).eq.'cationic')MconstB(ch-1)=1.0
if(Ty(2).eq.'anionic ')MconstB(ch-1)=2.0
if(Ty(2).eq.'nonionic')MconstB(ch-1)=3.0
     READ (13,507)
            READ (13, 508) T
 508 FORMAT (F15.0)
     READ (13,507)
 509 FORMAT (32X,F15.0)
 510 FORMAT (45X,F15.0)
 511 FORMAT (58X,F15.0)
 512 FORMAT (22X,F15.0)
 513 FORMAT (80X,F15.0)
            READ (13, 509) MconstA(1)
            MconstA(2)=0.0
            MconstA(3)=0.0
            MconstA(4)=0.0
            READ (13, 510) MconstA(5)
            READ (13, 511) MconstA(6)
            READ (13, 512) MconstA(7)
            READ (13, 513) MconstA(9)
  READ (13,507)
            READ (13, 509) MconstB(1)
            MconstB(2)=0.0
            MconstB(3)=0.0
            MconstB(4)=0.0
            READ (13, 510) MconstB(5)
            READ (13, 511) MconstB(6)
            READ (13, 512) MconstB(7)
            READ (13, 513) MconstB(9)
  READ (13,507)
                        READ (13, 514) CsurA,CsurB,Cadd
 514 FORMAT (3(E15.0/))
      CLOSE(13)
end
!**********************************************************************
```

```
subroutine shape_distrib
implicit none
real*8 X1A,X1B,alfaA1,cylopt(1,2),sphopt(1,2),capopt(1,2),gg(1,2)
real*8 chpot,chpotcyl,g_sphNN
real*8 alfa, Nmin, Nmax, nn, nshag, Xg, Rleft,Rright,RUbound, vs, cff,Rsphopt,Rcap,Ncap, alfacap, gcap,alfacyl,Rcyl,gcyl,grod,Y,wwju
real*8 r_patch,alfa_pl, g_p, alfa_jun,bjun,cjun,g_j, b,c, rat, gjun, g_jun2, g_tor, n_tor,n_pl, njun,alfplane,alfa_aver,bpl,v_tor_b, v_patch
real*8 lsinf,lssup
common /lsformin/lsinf,lssup
COMMON /alfabound/right_alfa,left_alfa
REAL*8 right_alfa,left_alfa


common /monomers/ X1A,X1B,alfaA1
common /geomopt/ cylopt,capopt,sphopt,Y
common /optpl/ r_patch,alfa_pl, g_p
common /optjun/ alfa_jun, bjun,cjun,g_j


OPEN(UNIT=19,FILE='Distrib_sphANDcyl.txt')
OPEN(UNIT=20,FILE='Distrib_junc.txt')

!spheres
Rsphopt = sphopt(1,1)
alfa = sphopt(1,2)
gg(1,2)=alfa
Rright = RUbound(alfa)
Rleft = 0.2d0*Rright                                    ! 0.2d0*Rright
cff = 4./3.*3.1415926/vs(alfa)
Nmax = cff*Rright**3
Nmin = cff*Rleft**3
do nn=Nmin,Nmax, 1.0d0
gg(1,1)=(nn/cff)**0.333333333
g_sphNN= chpot(gg)
if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15)then
Xg = (X1A*alfa+X1B*(1.d0 - alfa))*dexp(- g_sphNN)
else
Xg = (X1A**(alfa))*(X1B**((1.0d0 - alfa)))*dexp(- g_sphNN)
endif
Xg=Xg**nn
write (19,'(5(2X,E12.6))') nn, Xg, nn*Xg, Xg*nn**2,g_sphNN
enddo
```

```
! cylinders
Rcap =   capopt(1,1)
alfacap =  capopt(1,2)
cff = 4./3.*3.1415926/vs(alfacap)
Ncap = cff*Rcap**3
print *, 'Ncap= ', Ncap
gg=capopt
gcap= chpot(gg)
Rcyl = cylopt(1,1)
alfacyl=cylopt(1,2)
gg = cylopt
gcyl = chpotcyl(gg)
nn=Ncap
nshag=1.0d0
do while (nn<1.d4)
alfa = alfacyl +Ncap*(alfacap-alfacyl)/nn
grod = gcyl +Ncap*(gcap-gcyl)/nn

if (alfa<1.0d-15.or.dabs(alfa-1.0d0)<1.0d-15)then
Xg = (X1A*alfa+X1B*(1.d0 - alfa))*dexp(-grod)
else
Xg = (X1A**alfa)*(X1B**(1.0d0 - alfa))*dexp(-grod)
endif
Xg=Xg**nn
write (19,'(6(2X,E12.6))') nn,Xg, nn*Xg, Xg*nn**2,grod,gcyl
nshag = 1.5d0*nshag
nn=nn+nshag
enddo

! junctions
alfa=alfa_jun                                    ! this is actually   alfa  for toroid part in the optimal junction
b=bjun
                                ! this is b toroid for an optimal junction
bpl=r_patch
                                ! this is b for the plane in the optimal junction  (optimal plane in this version)
alfplane = alfa_pl
            !          this is alfa for the plane in the optimal junction  (optimal plane in this version)
do rat=1.5d0, 250.0d0, 0.5d0
c=rat*b
n_tor =    v_tor_b(b,c)/vs(alfa)
n_pl =    v_patch(bpl,c)/vs(alfplane)
njun = n_tor+n_pl
gjun = (n_tor*g_tor(b,c,alfa)+n_pl*g_p)/njun
alfa_aver = (n_tor*alfa + n_pl*alfplane)/njun
```

```
      nn = njun
      if (alfa_aver<1.0d-15.or.dabs(alfa_aver-1.0d0)<1.0d-15)then                    !  may be here and elsewhere it is better to use  alfaA1  an place of alfa  when processing the pure component
      Xg = (X1A*alfa_aver+X1B*(1.d0 - alfa_aver))*dexp(-gjun)
      wwju = dlog(X1A*alfa_aver+X1B*(1.d0 - alfa_aver))
      else
      Xg = (X1A**alfa_aver)*(X1B**(1.0d0 - alfa_aver))*dexp(-gjun)
      wwju = alfa_aver*dlog(X1A)+       (1.0d0 - alfa_aver)*dlog(X1B)
      endif
      if (Xg<1.0d0)then
      Xg=Xg**nn
      else
      Xg=1.0d0
      endif
      write                    (20,'(12(2X,E12.6))')                    nn,Xg,                    nn*Xg, Xg*nn**2,gjun,g_jun2(b,c,alfa),n_tor,n_pl,alfa_aver,alfa,alfplane,nn*(wwju-gjun)
      enddo
      close(19)
      close(20)
      return
      end
      !**********************************************************************
      subroutine micelles(X1,Ybig1,micellesgXg)
      implicit none
      real*8 X1A,X1B,alfaA1,X1
      real*8 njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun
      real*8 gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
      common /monomers/ X1A,X1B,alfaA1
      real*8  lsinf,lssup
      logical Ybig1
      common /lsformin/ lsinf,lssup
      real*8 Xzero(1,2),S0,S1,S2,XArod,XBrod,nsph,gXgsph,g2Xgsph,micellesgXg,Xgsph,XAsph,XBsph
      common /exitvalue/ S0,S1,S2,XArod, nsph,gXgsph,g2Xgsph,Xgsph,XAsph,XBsph,XBrod
      common/exitjun/ njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun
      common/exitves/gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
      X1A = X1*alfaA1
      X1B = X1*(1.0d0 - alfaA1)
      !!!!!!!!!!!!!!!!!!!!!!!!!!Xzero(1,1) = lsinf
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Xzero(1,1) = lssup*0.8
      Xzero(1,1) = 0.5*lsinf+0.5*lssup
      Xzero(1,2) = 0.5d0

      call gXgrod(Xzero,S0,S1,S2,XArod,XBrod,nsph,Ybig1)
      if (Ybig1 .eqv..true.) return
```

```
call gXgsphsub(nsph,gXgsph,g2Xgsph,Xgsph,XAsph,XBsph,Ybig1)
if (Ybig1 .eqv..true.) return
gXg_jun = 0.0d0
!!call gXgjunsub(njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun,Ybig1)
!!if (Ybig1 .eqv..true.) return
call gXgvessub(gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves,Ybig1)
if (Ybig1 .eqv..true.) return
micellesgXg = S1 + gXgsph + gXg_jun+gXg_ves
return
end
!**********************************************************************
subroutine gXgvessub(gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves,Ybig1)
implicit none
real*8 X1A,X1B,alfaA1
real*8 nves !,gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
logical Ybig1
common /monomers/X1A,X1B,alfaA1
real*8 lsinf,lssup,ls_min
common /lsformin/lsinf,lssup
COMMON /alfabound/right_alfa,left_alfa
REAL*8 xguess(3), x(3), ftol, s, g_v,g_ves,chpot,aINT,bINT
double precision sg1,sg2,s1,sA,sB,gXg_ves,g2Xg_ves,Xg_ves,XA_ves,XB_ves
REAL*8 right_alfa,left_alfa,fves !,s1,s2,sA,sB
real*8 nvesi,nveso,agg_ves,agg_vesi,agg_veso,n
common /border_ves/aINT,bINT
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
!   1     2     3     4     5     6     7     8              9
!  Nc     Vs    ls    a0    ap   delta deltaH  logic   ddip
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
    nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
!external fves
REAL*8 a_veso,a_vesi,a_vi,a_vo
if ((alfaA1.LE.1.0d-10).or.(alfaA1.GE.1.0d0-1.0d-14)) return
Ybig1 = .false.
sg1=0.0d0
ls_min=lsinf/0.3d0
aINT=12.0d0*ls_min
bINT=101.01*ls_min
call qtrap(aINT,bINT,sg1,sg2,s1,sA,sB)        !integral over r_ves to find gXg_ves and others
if(s1.eq.100) then
  Ybig1 = .true.
```

```
 return
end if
    gXg_ves = sg1
            g2Xg_ves = sg2
            Xg_ves =s1
            XA_ves =sA
            XB_ves = sB
return
end subroutine
!**********************************************************************
subroutine gXgjunsub(njun,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun,Ybig1)
implicit none
real*8 X1A,X1B,fcyl,fsph,alfaA1
real*8 n,njun,n_tor,n_pl,gXg_jun,g2Xg_jun,Xg_jun,XA_jun,XB_jun,vs,c_jun,v_patch,v_tor_b,g_jun2
external fcyl,fsph
logical Ybig1
common /monomers/ X1A,X1B,alfaA1
real*8 lsinf,lssup,bleft,bright,Rubound
common /lsformin/ lsinf,lssup
COMMON/alfabound/right_alfa,left_alfa
real*8 gcylopt,alfacylopt,Rcylopt,alfasphopt,Rsphopt,gsphopt
REAL*8 xguess(4), x(4), ftol, s, g_p, g_j
REAL*8 b, bjun, cjun, r_patch, alfa,r_pl,alfa_pl,alfa_jun,Xgiteration
REAL*8 right_alfa,left_alfa, fcn_pl,g_pl,fcn_jun
integer depend
    COMMON/control/depend
common /optcyl/ alfacylopt,Rcylopt,gcylopt
common /optsph/ alfasphopt,Rsphopt,gsphopt
common /optpl/ r_patch,alfa_pl, g_p
common /optjun/ alfa_jun, bjun,cjun,g_j
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
!   1     2     3     4     5     6     7     8              9
!  Nc     Vs    ls    a0    ap   delta deltaH  logic   ddip
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
   nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
external fcn_pl,fcn_jun, fcn_pl_pure, fcn_jun_pure , fcn_junmax         , fcn_jun_pureMAX

Ybig1 = .false.
b=Rcylopt                                                    !initial values as in opt cylinder
!    PURE COMPONENT CALCULATIONS
if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15)then
```

```
bright=RUbound(alfaA1)
bleft=0.3d0*bright
  ! changed from 0.2
xguess(1) = Rcylopt/bright
x(1)= xguess(1)                                    !  when optimization is turned off
     s=1.0d-7
     ftol=1.0d-9
          CALL dumpol(fcn_pl_pure,1,xguess,s,ftol,1000,x,g_p)
          r_pl=x(1)*bright
          alfa_pl=alfaA1
     if (r_pl.LT.bleft) r_pl=bleft
           if (r_pl.GT.bright) r_pl=bright
g_p=g_pl(r_pl,alfa_pl)
r_patch=r_pl                                       !  r_patch is an optimal parameter for the plane
! optimization for junction
     depend=0                !0 means true. We use g_jun2 instead of g_jun
          xguess(1) = Rcylopt/bright
          xguess(2) = xguess(1)*10.00d0
          s=0.003d0
          ftol=1.0d-9
          CALL dumpol(fcn_jun_pureMAX,2,xguess,s,ftol,10000,x,g_j)
          bjun=x(1)*bright
          cjun=x(2)*bright
          alfa_jun=alfaA1
          if (bjun.LT.bleft) bjun=bleft
          if (bjun.GT.bright) bjun=bright
          if ((cjun/bjun).LT.1.10d0) cjun=1.10d0*bjun
          if ((cjun/bjun).GT.100.0d0) cjun=100.0d0*bjun
          g_j=g_jun2(bjun,cjun,alfa_jun)
alfa=alfa_jun
njun = (v_tor_b(bjun,cjun)+v_patch(r_patch,c_jun))/vs(alfa_jun)
n=njun
Xgiteration = (X1A*alfa+X1B*(1.d0 - alfa))*dexp(- g_j)
!  XA1  or XB1  for pure component, depending on alfa
if(Xgiteration>=1.) then
  Ybig1 = .true.
  return
end if
           Xgiteration = Xgiteration**n
    gXg_jun = n*Xgiteration
           g2Xg_jun = Xgiteration*n**2
           Xg_jun = Xgiteration
           XA_jun = n*alfa*Xgiteration
           XB_jun = n*(1.d0-alfa)*Xgiteration
```

```
return
endif                                          ! end of calculation for pure components
!     CALCULATION  FOR MIXTURES
alfa=alfacylopt
! optimization for plane
left_alfa = 1.0d-14              !0.0d0
right_alfa = 1.0d0-          1.0d-14                    !1.0d0

xguess(1) = Rcylopt/lssup
xguess(2) = alfacylopt
x(1)= xguess(1)                                         !  when optimization is turned off
x(2)= xguess(2)
     s=1.0d-7
     ftol=1.0d-9
          CALL dumpol(fcn_pl,2,xguess,s,ftol,1000,x,g_p)
          r_pl=x(1)*lssup
          alfa_pl=x(2)
!write(*,*) 'planeYYY',r_pl,alfa_pl
     if (r_pl.LT.lsinf) r_pl=lsinf
           if (r_pl.GT.lssup) r_pl=lssup
          if (alfa_pl.LT.left_alfa) alfa_pl=left_alfa
          if (alfa_pl.GT.right_alfa) alfa_pl=right_alfa
g_p=g_pl(r_pl,alfa_pl)
r_patch=r_pl                                            !  r_patch is an optimal parameter for the plane

! optimization for junction
     depend=0                    !0 means true. We use g_jun2 instead of g_jun
          xguess(1) = Rcylopt/lssup
          xguess(2) = xguess(1)*10.00d0
          xguess(3) = alfacylopt
          s=0.003d0
          ftol=1.0d-9
          CALL dumpol(fcn_junMAX,3,xguess,s,ftol,10000,x,g_j)
                    !            fcn_jun   -minimizes free energy
!write(*,*) 'junction',g_j,x(1),x(2),x(3)
          bjun=x(1)*lssup
          cjun=x(2)*lssup
          alfa_jun=x(3)
          if (bjun.LT.lsinf) bjun=lsinf
          if (bjun.GT.lssup) bjun=lssup
          if (alfa_jun.LT.left_alfa) alfa_jun=left_alfa
          if (alfa_jun.GT.right_alfa) alfa_jun=right_alfa
          if ((cjun/bjun).LT.1.10d0) cjun=1.10d0*bjun
          if ((cjun/bjun).GT.100.0d0) cjun=100.0d0*bjun
```

```
            g_j=g_jun2(bjun,cjun,alfa_jun)
n_tor =    v_tor_b(bjun,cjun)/vs(alfa_jun)
n_pl =    v_patch(r_patch,c_jun)/vs(alfa_pl)
njun = n_tor+n_pl
alfa = (n_tor*alfa_jun + n_pl*alfa_pl)/njun
n=njun
Xgiteration = (X1A**(alfa))*(X1B**((1.d0 - alfa)))*dexp(- g_j)

if(Xgiteration>=1.) then
  Ybig1 = .true.
  return
end if
            Xgiteration = Xgiteration**n
     gXg_jun = n*Xgiteration
            g2Xg_jun = Xgiteration*n**2
            Xg_jun = Xgiteration
            XA_jun = n*alfa*Xgiteration
            XB_jun = n*(1.d0-alfa)*Xgiteration
return
end subroutine

!**********************************************************************
subroutine gXgrod(Xzero,S0,S1,S2,XArod,Xbrod,nsph,Ybig1)
implicit none
real*8 X1A,X1B,Xzero(1,2),fcyl,fsph,XXeps(1,2),chpotcyl
external fcyl,fsph
common /monomers/ X1A,X1B,alfaA1
real*8 S0,S1,S2,XArod,XBrod,nsph,minX(1,2), minZ,Y,V,VV
logical Ybig1
real*8 lsinf,lssup
common /lsformin/ lsinf,lssup
real*8 gcylopt,alfacylopt,Rcylopt,alfasphopt,Rsphopt,gsphopt
common /optcyl/ alfacylopt,Rcylopt,gcylopt
common /optsph/ alfasphopt,Rsphopt,gsphopt
real*8  cylopt(1,2),capopt(1,2),sphopt(1,2),chpot,J,lnK,alfaA1,vs
common /geomopt/ cylopt,capopt,sphopt,Y
real*8    XX(1,2),logicalcoord,minxx,minyy,bleft,bright,RUbound
common /coordsquarecom / logicalcoord,XX
data XXeps/0.001,0.005/
external chpot
Ybig1 = .false.

if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15)then
logicalcoord=2.
```

```
XX(1,2)=alfaA1
bright=RUbound(alfaA1)
bleft =0.3d0*bright                                                    ! changed from 0.2
call golden(fcyl,bleft,bright,0.1d-04,minxx,minyy)
alfacylopt=alfaA1
Rcylopt=minxx
minX(1,1)=Rcylopt
minX(1,2)=alfacylopt
cylopt = minX
gcylopt = chpotcyl(minX)
Y = (X1A*alfacylopt+X1B*(1.d0 - alfacylopt))*dexp(-gcylopt)          ! when alfaA1=0 or 1 gives X1B or X1A, respectively
if(Y>=1.) then
  Ybig1 = .true.
  return
end if
call golden(fsph,bleft,bright,0.1d-04,minxx,minyy)
Rsphopt=minxx
alfasphopt=alfaA1
minX(1,1)= Rsphopt
minX(1,2)=alfasphopt
sphopt = minX
capopt = minX
gsphopt=chpot(minX)
nsph = 4./3.*3.1415926*Rsphopt**3/vs(alfasphopt)
lnK = nsph*(gsphopt - gcylopt)
VV = -lnK+nsph*dlog(Y)
if (VV<0) then
S0 = dexp(VV)/(1.0d0 - Y)
S1 = (nsph + Y/(1.0d0 - Y))*S0
 ! S1rod   Sum of all  (nA+nB)*XnAnB   for cylinders
S2 = (nsph + Y/(1.0d0 - Y))*S1 + Y/(1.0d0 - Y)*(1.0d0 + Y/(1.0d0 - Y))*S0    ! Sum of all (nA+nB)^2*XnAnB  (misprint in Vasya's  eq.2.4.12)
XArod = alfacylopt*S1
XBrod = (1.0d0-alfacylopt)*S1
else
Ybig1 = .true.
endif
return
endif   !  end of pure component part
call  coordsquare(fcyl,Xzero,lsinf,lssup,0.1d-04,1.0d-14,(1.d0-1.0d-14),0.3d-7,XXeps,minX,minZ)
alfacylopt = minX(1,2)
Rcylopt = minX(1,1)
cylopt = minX
```

```
gcylopt = chpotcyl(minX)
Y = (X1A**alfacylopt)*(X1B**(1. - alfacylopt))*dexp(-gcylopt)
if(Y>=1.) then
  Ybig1 = .true.
  return
end if
!***************probe:
call  coordsquare(fsph,Xzero,lsinf,lssup,0.1d-04,1.0d-14,(1.d0-1.0d-14),0.3d-7,XXeps,minX,minZ)

sphopt = minX
capopt = minX
alfasphopt=minX(1,2)
Rsphopt= minX(1,1)
gsphopt=chpot(minX)
nsph = 4./3.*3.1415926*Rsphopt**3/vs(alfasphopt)
lnK = nsph*(gsphopt - gcylopt)

J = nsph*(alfasphopt - alfacylopt)
VV = J*dlog(X1A/X1B)-lnK+nsph*dlog(Y)
if (VV<0) then                  !       check whether the fraction of cylinders is < 1
S0 = dexp(VV)/(1.0d0 - Y)
S1 = (nsph + Y/(1.0d0 - Y))*S0
 !  S1rod   Sum of all  (nA+nB)*XnAnB   for cylinders
S2 = (nsph + Y/(1.0d0 - Y))*S1 + Y/(1.0d0 - Y)*(1.0d0 + Y/(1.0d0 - Y))*S0     !       Sum     of      all
(nA+nB)^2*XnAnB                 (misprint in Vasya's  eq.2.4.12)
XArod = alfacylopt*S1 + J*S0
XBrod = (1.0d0-alfacylopt)*S1-J*S0
endif
return
end subroutine
!*********************************************************************
subroutine gXgsphsub(nsph,gXgsph,g2Xgsph,Xgsph,XAsph,XBsph,Ybig1)
implicit none
logical Ybig1
real*8 nsph,gXgsph,g2Xgsph,alfa,chag,n,nsphint,Xgsph,Xgiteration,XAsph,XBsph
real*8 vs,chpot,alfaA1,fsph_max,fsph_gmax, Xzero(1,2),XXeps(1,2),minX(1,2), minZ
integer nsphere, nn
real*8 X1A,X1B,alfasphopt,Rsphopt,gsphopt, R,gg(1,2)
real*8 lsinf,lssup
real*8    XX(1,2),logicalcoord,minxx,minyy,bleft,bright,RUbound
real*8  cylopt(1,2),sphopt(1,2),capopt(1,2),Y
common /geomopt/ cylopt,capopt,sphopt,Y
common /coordsquarecom / logicalcoord,XX
common /lsformin/ lsinf,lssup
```

```
common /monomers/ X1A,X1B,alfaA1
common /optsph/ alfasphopt,Rsphopt,gsphopt
data XXeps/0.001,0.005/
external chpot,fsph_max,fsph_gmax
Ybig1 = .false.
XAsph = 0.
XBsph = 0.
Xgsph = 0.
gXgsph = 0.
g2Xgsph = 0.
if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15) then                    ! pure component A
or B
alfa = alfaA1
logicalcoord=2.
XX(1,2)=alfa
bright=RUbound(alfa)
bleft =0.4d0*bright
call golden(fsph_gmax,bleft,bright,0.1d-04,minxx,minyy)
Rsphopt=minxx
alfasphopt=alfa
minX(1,1)= Rsphopt
minX(1,2)=alfasphopt
sphopt = minX
gsphopt=chpot(minX)
  Xgiteration = (X1A*alfa+X1B*(1.d0 - alfa))*dexp(-gsphopt)
if(Xgiteration>=1.) then
 Ybig1 = .true.
 return
end if
nsph = 4./3.*3.1415926*Rsphopt**3/vs(alfa)
write(*,*)'sph=',nsph,Xgiteration
n=nsph
        Xgiteration = Xgiteration**n
   gXgsph = gXgsph +  n*Xgiteration
        g2Xgsph = g2Xgsph + Xgiteration*n**2
        Xgsph = Xgsph + Xgiteration
        XAsph = XAsph + n*alfa*Xgiteration
        XBsph = XBsph + n*(1.d0-alfa)*Xgiteration
return
endif
!                MIXTURES    A+B
Xzero(1,1) = Rsphopt
Xzero(1,2) = alfasphopt
call  coordsquare(fsph_gmax,Xzero,lsinf,lssup,0.1d-04,1.0d-14,(1.d0-1.0d-14),0.3d-7,XXeps,minX,minZ)
```

```
      alfasphopt=minX(1,2)
      Rsphopt= minX(1,1)
      sphopt = minX
      gsphopt=chpot(minX)
      alfa =alfasphopt
      nsph = 4./3.*3.1415926*Rsphopt**3/vs(alfa)                    !  spheres present at maximum concentration at current monomer fraction
      n =  nsph                                                     !          if we prefer real aggr numbers
        R = (3.0d0/4./3.1415926*n*vs(alfa))**0.3333333
            gg(1,1) = R
            gg(1,2) = alfa
            Xgiteration = (X1A**(alfa))*(X1B**((1.0d0 - alfa)))*dexp(- chpot(gg))

      if(Xgiteration>=1.) then
       Ybig1 = .true.
       return
      end if
            Xgiteration = Xgiteration**n
        gXgsph = gXgsph +  n*Xgiteration
            g2Xgsph = g2Xgsph + Xgiteration*n**2
            Xgsph = Xgsph + Xgiteration
            XAsph = XAsph + n*alfa*Xgiteration
            XBsph = XBsph + n*(1.d0-alfa)*Xgiteration
      return
      end subroutine
      !*********************************************************************** subroutine CMC (X1,Ybig1,CMCfun)
      implicit none
      real*8 X1,CMCfun,micellesgXg
      real*8 X1A,X1B,alfaA1,XtotA,XtotB
      logical Ybig1
      common/gross_mfrac/XtotA,XtotB
      common /monomers/ X1A,X1B,alfaA1

      call micelles(X1,Ybig1,micellesgXg)
      if(micellesgXg>=1.2d0)    Ybig1 = .true.
      print *,'from CMC ',Ybig1,micellesgXg, X1
      if (Ybig1 .eqv..true.) return
      if(XtotA==0.and.XtotB==0.)then
      CMCfun = (micellesgXg - X1)**2./X1**3.    !  THIS FUNCTION IS USED FOR   CMC-calculation
      else
      CMCfun = (XtotA+XtotB-micellesgXg - X1)**2./X1**3.         ! Material  Balance
      !CMCfun = (0.080*0.018-micellesgXg - X1)**2./X1**3.  !(0.025*0.018 - X1 - micellesgXg)**2.
      endif
```

```
return
end subroutine
!**********************integrals for vesicles ********************
subroutine trapzd(aINT,bINT,s1,n,s2,s,sA,sB)          ! to make value of integral more precise
INTEGER n
double precision aINT,bINT,s,fgXgves,s1,s2,sA,sB,t,tA,tB,A,B,told
real*8 nves,nA,nB,nX,alfa,alfaA,alfaB,alfaX,alfaXold,nXold
common /integral/nves,alfa
EXTERNAL fgXgves
INTEGER it,counter
double precision del,sum,tnm,x,sum2,sum1,sumA,sumB,delN

if (n.eq.1) then
        tA=fgXgves(aINT,bINT)
        nA=nves                                          !Aint
        A=nA
        alfaA=alfa
        if      (tA.ge.1.0d0) then
        s1=100
        return
        end if

        tB=fgXgves(bINT,bINT)
        nB=nves                                            !Bint
        B=nB
        alfaB=alfa
        if      (tB.ge.1.0d0) then
        s1=100
        return
        end if

        s1=0.5d0*(B-A)*(tA+tB)
  s=0.5d0*(B-A)*(tA/nA+tB/nB)
  s2=0.5d0*(B-A)*(nA*tA+nB*tB)
  sA=0.5d0*(B-A)*(alfaA*tA+alfaB*tB)
  sB=0.5d0*(B-A)*((1-alfaA)*tA+(1-alfaB)*tB)
else
        it=2**(n-2)
        tnm=it
        del=(bINT-aINT)/tnm                    !deltaR
        x=aINT+0.5d0*del                       !Ri
        sum1=0.0d0
        sum=0.0d0
        sum2=0.0d0
```

```
            sumA=0.0d0
            sumB=0.0d0
            do counter=1,it
            t=fgXgves(x,bINT)                                   !ti
            nX=nves                                                 !ni
            alfaX=alfa



            if          (t.ge.1.0d0) then
            s1=100
            return
            end if
            if (counter.eq.1) then
            nXold=nX                                            !ni-1
            told=t                                              !ti-1
            alfaXold=alfaX
            else                                                !deltaN/2
  delN=0.5d0*(nX-nXold)
  sum1=sum1+(t+told)*delN
  sum=sum+(t/nX+told/nXold)*delN
  sum2=sum2+(nX*t+told*nXold)*delN
            sumA=sumA+(alfaX*t+told*alfaXold)*delN
            sumB=sumB+((1-alfaX)*t+told*(1-alfaXold))*delN
            x=x+del
            nXold=nX                                            !to create ni-1
            told=t                                              !ti-1
            alfaXold=alfaX
            endif
            enddo
        s1=sum1
            s=sum
            s2=sum2
            sA=sumA
            sB=sumB
            if          (s1.ge.1.0d0) then
            s1=100
            return
            end if
      endif
      return
      END
      !**********************************************************************
      subroutine qtrap(aINT,bINT,s1,s2,s,sA,sB)    !integrals to find gXgves (s),g2Xgves (s2),gXAves (sA),gXBves
(sB)
```

```
integer JMAX
double precision aINT,bINT,fgXgves,s,EPS,s2,sA,sB,s1

EXTERNAL fgXgves
PARAMETER (EPS=1.e-1,JMAX=15)
integer n
double precision olds1
olds1=-1.e-30
          do n=1,JMAX
          call trapzd(aINT,bINT,s1,n,s2,s,sA,sB)
          if (s1.eq.100.0d0) then
   return
   endif

          if (n.gt.5) then
           if (s1.le.1.0d-50) then
                    if (dabs(s1-olds1).lt.2*EPS*dabs(olds1).or.(s1.le.1.0d-100)) then
                    return
                    end if
            else
                   if (dabs(s1-olds1).lt.EPS*dabs(olds1).or.(s1.le.1.0d-100)) then
                   return
                   end if
    end if
          end if
          olds1=s1
          end do
          pause "too many steps in qtrap"
return
END
!*****************************************************************
double precision function fgXgves(a,bINT)
IMPLICIT none
Common /ves_dumpol/r_ves
REAL*8 r_ves,tauo,taui,alfao,alfai,alfa,nveso,nvesi,nves,a,b,bINT,vs
common /monomers/ X1A,X1B,alfaA1
real*8 lsinf,lssup,agg_vesi,agg_veso,agg_ves,X1A,X1B,alfaA1,n
common /lsformin/ lsinf,lssup
common /integral/ nves,alfa
double precision   Xgiteration
REAL*8 xguess(3), x(3), ftol, s, g_v,g_ves,chpot,fves,ls_min,f2ves,fmin,C
external fves,f2ves
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch),chpotmin,alfaimin,alfaomin,tauimin,tauomin
```

```
REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB, &
epsilon,MconstA, MconstB, &
nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
Common /ves_dumpol1/alfao,alfai
Common /tauotaui/C
integer i
r_ves=a
b=bINT    !not used
ls_min=lsinf/0.3d0
 fmin=1000
 chpotmin=1000
  do alfao=0.25d0,0.65d0,0.02d0
  do alfai=0.25d0,0.65d0,0.02d0
xguess(1)=1.182                    !for tau=tauo+taui  !0.7 for Cmim+SDBS
s=1.0d-7
ftol=5.0d-11
C=0.9d0
CALL dumpol(f2ves,1,xguess,s,ftol,10000,x,g_v)
            tauo=x(1)*ls_min/(1+C)
            taui=C*tauo

            chpot=g_ves(r_ves,tauo,taui,alfao,alfai)
            nvesi=agg_vesi(r_ves,taui,alfai)
nveso=agg_veso(r_ves,tauo,taui,alfao)
nves=agg_ves(r_ves,tauo,taui,alfao,alfai)
alfa = (alfao*nveso+ alfai*nvesi)/nves
g_v=g_ves(r_ves,tauo,taui,alfao,alfai)-alfa*dlog(X1A)-(1.0d0-alfa)*dlog(X1B)
if (g_v.LT.fmin) then
            fmin=g_v
            chpotmin=chpot
            alfaomin=alfao
            alfaimin=alfai
            tauomin=tauo
            tauimin=taui
            endif
            enddo
            enddo
            chpot=chpotmin
            alfao=alfaomin
            alfai=alfaimin
            tauo=tauomin
            taui=tauimin
```

```
nvesi=agg_vesi(r_ves,taui,alfai)
nveso=agg_veso(r_ves,tauo,taui,alfao)
nves=agg_ves(r_ves,tauo,taui,alfao,alfai)
alfa = (alfao*nveso+ alfai*nvesi)/nves

g_v=g_ves(r_ves,tauo,taui,alfao,alfai)-alfa*dlog(X1A)-(1.0d0-alfa)*dlog(X1B)
!chpot=g_ves(r_ves,tauo,taui,alfao,alfai)
n=nves
Xgiteration = (X1A**(alfa))*(X1B**(1.-alfa))*dexp(- chpot)
if(Xgiteration>=1.) then
 fgXgves =100          !Ybig1=true
 return
end if
          fgXgves=dlog(n)+n*dlog(Xgiteration)
          fgXgves = dexp(fgXgves)
RETURN
END
! ********** Main Functions: chemical potential of sphere ********************
real*8 function chpot (gg)
implicit none
REAL*8 StericFrEn,DefFrEn,TransferFrEn, g_ion_nlin, g_ion, &   ! chem potential of spherical mocelle
   InterfacialFrEn,Ohshima_g_diff,MixFrEn, g_def
real*8 gg(1,2)
integer Ldamp
real*8 H,CH2LGTH,VTAILA,VTAILB,VHEADA,VHEADB, gsh
REAL*8 a,Cadd,P,Rs,X1A,X1B,Vs,alfagA,DifStandChPot,alfaA1,g_zwitter
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)   ! surfactant molecular parameters:
!  1    2    3    4    5    6    7     8       9
!  Nc    Vs    ls    a0    ap   delta deltaH  logic     ddip
! logic =  1 - cationic surfactant, 2 - anionic surfactant, 3 - nonionic surfactant: ddip<0   -   nonpolar;   ddip>0   -
dipole length for zwittwrionic surfactant
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
   nzero,krDeb,dchpotCH2,dchpotCH3,CsurA,CsurB
CHARACTER*3 shape
CHARACTER*8 MDL
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
 COMMON /Y/ H,VheadA,VheadB,VtailA,VtailB,CH2lgth
 COMMON /word/ MDL
real*8 tip ! tip = 1 - вычисления для сфер, 2 - вычисления для цилиндров
common /tip/ tip
common /monomers/ X1A,X1B,alfaA1
common /out/ ldamp
```

```
real*8 alfagion, delta, adelta,s,x0, a_sph, achrg
tip = 1.
alfagA = gg(1,2)
Rs = gg(1,1)

a = a_sph(Rs,alfagA)
P = vs(alfagA)/a/Rs

if(Mdl.eq.'new') then                                   ! new model - complex corona structure
shape = 'sph'
DifStandChPot = DefFrEn(Rs,P,alfagA) + TransferFrEn(alfagA) + MixFrEn(alfagA) +&
InterfacialFrEn(a,alfagA)
call shell (shape,alfagA,Rs,gsh)
if (MconstA(8)/=3.or.MconstB(8)/=3.) then
        call alf_ion_Deb(alfagA,alfagion,delta)          ! calculate ionic composition of the micelle and Debye length at current monomer fractions of ionic surfactants
  achrg=adelta(2.0d0,alfagA,Rs+H,0.0,delta)              ! area per molecule at Rs+H+delta
        s =4*Pi*alfagion*e**2/(epsilon*krDeb*achrg*(1.0d-16)*k*T)
        x0 = krDeb*(Rs + H + delta)*1.d-8
 DifStandChPot = DifStandChPot + Ohshima_g_diff(2.0d0, x0, s)
end if
chpot = gsh + DifStandChPot
if (ldamp.GT.0) write (11,'(2X,F8.4,8(2X,E12.5))') alfagA,DefFrEn(Rs,P,alfagA),TransferFrEn(alfagA),MixFrEn(alfagA),&
InterfacialFrEn(a,alfagA),Ohshima_g_diff(2.0d0, x0, s),chpot,Rs,H
return
end if

if (MconstA(8)/=3.or.MconstB(8)/=3.) then
                                                ! free energy of ionic micelles
        DifStandChPot = StericFrEn(a,alfagA) + g_def(Rs,0.0d0,alfagA,2.0d0) + TransferFrEn(alfagA) +&
                        ! g_def(Rs,0.0,alfagA,2.0d0) - def part in our model DefFrEn(Rs,P,alfagA) - Nagarajan def part
        InterfacialFrEn(a,alfagA) + MixFrEn(alfagA) + g_ion(Rs,0.0d0,alfagA,2.0d0)  ! g_ion(Rs,0.0d0,alfagA,2.0d0) g_ion_nlin(alfagA,Rs,0.0,2.0d0) Ohshima_g_diff(2.0d0, x0, s) Electrostatics
else
        DifStandChPot = StericFrEn(a,alfagA) + DefFrEn(Rs,P,alfagA) + TransferFrEn(alfagA) +&
  ! free energy og nonionic micelles with deformation part as in Nagarajan model
        InterfacialFrEn(a,alfagA) + MixFrEn(alfagA)
end if
! ADD DIPOLAR PART of FREE ENERGY
if (MconstA(9).GT.0.or.MconstB(9).GT.0.) then           ! if at least one surfactant is zwitterionic
  DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rs,0.0d0,2.0d0)
elseif (MconstA(8)*MconstB(8)==2.)then                  ! cationic + anionic OR cationic+anionic
```

```
      DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rs,0.0d0,2.0d0)          !(alfa,b,c,sh)
    endif
    chpot = DifStandChPot

    if (ldamp.GT.0) write (11,'(2X,F8.4,10(2X,E12.5))')
alfagA,DefFrEn(Rs,P,alfagA),TransferFrEn(alfagA),MixFrEn(alfagA),&
    InterfacialFrEn(a,alfagA),StericFrEn(a,alfagA),g_zwitter(alfagA,Rs,0.0d0,2.0d0),
g_ion(Rs,0.0d0,alfagA,2.0d0),g_ion_nlin(alfagA,Rs,0.0d0,2.0d0),chpot,Rs

    return
    end   function
    !********************** ** Chem potential of a cylinder ******************
    real*8 function chpotcyl(gg)
    implicit none
    integer, parameter :: ch = 9
    real*8 MconstA(ch), MconstB(ch) ,gg(1,2),Rc,alfagA,a,P,DifStandChPot
    !  1    2    3    4    5    6    7    8          9
    !  Nc    Vs    ls    a0    ap   delta deltaH  logic  ddip
    real*8 StericFrEn,DefFrEn,TransferFrEn,InterfacialFrEn,MixFrEn,Ohshima_g_diff,g_ion_nlin
    REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,CsurA,CsurB,&
       nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,X1A,X1B,alfaA1
    CHARACTER*3 shape
    CHARACTER*8 MDL
     COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
         nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
     COMMON /Y/ H,VheadA,VheadB,VtailA,VtailB,CH2lgth
     COMMON /word/ MDL
     integer ldamp
    real*8 tip ! tip = 1 - вычисления для сфер, 2 - вычисления для цилиндров
    common /tip/ tip
    common /monomers/ X1A,X1B,alfaA1
    common /out/ ldamp
    real*8 alfagion, delta, adelta, s, x0
    real*8 H,GSH,CH2LGTH,VTAILB,VTAILA,VHEADA,VHEADB, achrg,a_cyl,g_ion,g_def,g_zwitter
    tip = 2.

    Rc = gg(1,1)
    alfagA = gg(1,2)
    a = a_cyl(Rc,alfagA)
    P = 0.5

    if(Mdl.eq.'new') then                                                    ! new model -
complex corona structure
    shape = 'cyl'
```

```
DifStandChPot = DefFrEn(Rc,P,alfagA) + TransferFrEn(alfagA) + MixFrEn(alfagA) +&
InterfacialFrEn(a,alfagA)
call shell (shape,alfagA,Rc,gsh)
if (MconstA(8)/=3.or.MconstB(8)/=3.) then
        call alf_ion_Deb(alfagA,alfagion,delta)                    ! calculate ionic composition of the micelle and Debye length at current monomer fractions of ionic surfactants
  achrg=adelta(1.0d0,alfagA,Rc+H,0.0,delta)                    ! area per molecule at Rc+H+delta
        s =4*Pi*alfagion*e**2/(epsilon*krDeb*achrg*(1.0d-16)*k*T)
        x0 = krDeb*(Rc + H + delta)*1.d-8
DifStandChPot = DifStandChPot + Ohshima_g_diff(1.0d0, x0, s)
end if
chpotcyl = gsh + DifStandChPot
if (ldamp.GT.0) write (14,'(2X,F8.4,8(2X,E12.5))') alfagA,DefFrEn(Rc,P,alfagA),TransferFrEn(alfagA),MixFrEn(alfagA),&
InterfacialFrEn(a,alfagA),Ohshima_g_diff(1.0d0, x0, s),chpotcyl,Rc,H
return
end if

if (MconstA(8)/=3.or.MconstB(8)/=3.) then
                                        ! free energy of ionic micelles
  DifStandChPot = StericFrEn(a,alfagA) + g_def(Rc,0.0d0,alfagA,1.0d0) + TransferFrEn(alfagA) +&
        ! DefFrEn(Rc,P,alfagA) - deformation in Nagarajan model  g_def(b,c,alfa,sh) - deformation in our model
  InterfacialFrEn(a,alfagA) + MixFrEn(alfagA) + g_ion(Rc,0.0d0,alfagA,1.0d0) !g_ion(Rc,0.0d0,alfagA,1.0d0)  g_ion_nlin(alfagA,Rc,0.0,1.0d0)  Ohshima_g_diff(1.0d0, x0, s)  Electrostatics
else
  DifStandChPot = StericFrEn(a,alfagA) + DefFrEn(Rc,P,alfagA) + TransferFrEn(alfagA) +&  ! free energy of nonionic micelles with deformation part as in Nagarajan model
  InterfacialFrEn(a,alfagA) + MixFrEn(alfagA)
end if
!
! ADD  DIPOLAR  PART  of FREE ENERGY
!
if (MconstA(9).GT.0.or.MconstB(9).GT.0.) then                    ! if at least one surfactant is zwitterionic
  DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rc,0.0d0,1.0d0)
elseif (MconstA(8)*MconstB(8)==2.)then                    ! cationic + anionic  OR  cationic+anionic
  DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rc,0.0d0,1.0d0)        !(alfa,b,c,sh)
endif
chpotcyl = DifStandChPot

if (ldamp.GT.0) write (14,'(2X,F8.4,10(2X,E12.5))') alfagA,DefFrEn(Rc,P,alfagA),TransferFrEn(alfagA),MixFrEn(alfagA),&
InterfacialFrEn(a,alfagA),StericFrEn(a,alfagA),g_zwitter(alfagA,Rc,0.0d0,1.0d0), g_ion(Rc,0.0d0,alfagA,1.0d0), g_ion_nlin(alfagA,Rc,0.0d0,1.0d0),chpotcyl,Rc
return
```

```
end function

!**********************************************************************
subroutine chpotves
implicit none
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
!   1     2     3     4     5     6     7     8              9
!  Nc     Vs    ls    a0    ap   delta deltaH  logic  ddip
real*8 StericFrEn,DefFrEn,TransferFrEn,InterfacialFrEn,DumMixFrEn,Ohshima_g_diff,g_ionves_nlin
real*8 ster,ion,ion_nlin,def,tr,int,mix
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,CsurA,CsurB,&
    nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,X1A,X1B,alfaA1
CHARACTER*3 shape
CHARACTER*8 MDL
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
      nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
 COMMON /Y/ H,VheadA,VheadB,VtailA,VtailB,CH2lgth
 COMMON /word/ MDL
 integer ldamp
!real*8 tip ! tip = 1 - вычисления для сфер, 2 - вычисления для цилиндров
!common /tip/ tip
common /monomers/ X1A,X1B,alfaA1
common /out/ ldamp
!real*8 alfagion, delta, adelta, s, x0
real*8 H,GSH,CH2LGTH,VTAILB,VTAILA,VHEADA,VHEADB, achrg,a_ves,g_ionves,g_def,g_zwitter
real*8 aINT,bINT, nves,nvesopt
real*8 lsinf,lssup,ls_min,alfaox,alfaix,C
common /lsformin/lsinf,lssup
COMMON /alfabound/right_alfa,left_alfa
REAL*8 xguess(3), x(3), ftol, s, g_v,g_ves,chpot
REAL*8 alfavesopt,alfaiopt,alfaoopt,Rvesopt,gvesopt,tauiopt,tauoopt
REAL*8 alfaves,alfai,alfao,r_ves,taui,tauo,Xgiteration,gvopt,fmin
REAL*8 right_alfa,left_alfa,f2ves,chpotmin,alfaimin,alfaomin,tauimin,tauomin
real*8 agg_vi,agg_vo,agg_ves,agg_vesi,agg_veso,n,a_veso,a_vesi,a_vo,a_vi,nveso,nvesi,nn
double precision Xg,Xit
common /optves/alfavesopt,alfaiopt,alfaoopt,Rvesopt,gvesopt,tauiopt,tauoopt,nvesopt
external fves,f2ves
common /border_rves/aINT,bINT
Common /ves_dumpol/r_ves
Common /ves_dumpol1/alfaox,alfaix
Common /tauotaui/C
integer i
OPEN(UNIT=31,FILE='Distrib_ves.txt')
```

```
write (31,*)'  Rves/ls        g          X          g*Xg        g2*Xg          chpot'    ! alfa     alfaA1'
left_alfa = 1.0d-14              !0.0d0
right_alfa = 1.0d0-1.0d-14       !1.0d0
gvopt=1000
ls_min=lsinf/0.3d0
aINT=2.0d0*ls_min
bINT=101.01*ls_min
do r_ves=aINT,bINT,0.5d0*ls_min
 fmin=1000
 do alfaox=0.25d0,0.65d0,0.02d0
  do alfaix=0.25d0,0.65d0,0.02d0
xguess(1)=1.182        !for tau=tauo+taui  ! 0.7  for Cmim+SDBS
s=1.0d-7
ftol=5.0d-11
C=0.9d0
CALL dumpol(f2ves,1,xguess,s,ftol,10000,x,g_v)
        tauo=x(1)*ls_min/(1+C)
        taui=C*tauo
        chpot=g_ves(r_ves,tauo,taui,alfaox,alfaix)
nvesi=agg_vesi(r_ves,taui,alfaix)
nveso=agg_veso(r_ves,tauo,taui,alfaox)
nves=agg_ves(r_ves,tauo,taui,alfaox,alfaix)
alfaves = (alfaox*nveso+ alfaix*nvesi)/nves
g_v=chpot-alfaves*dlog(X1A)-(1.0d0-alfaves)*dlog(X1B)
if (g_v.LT.gvopt) then
   gvopt=g_v                                          !optimal among all values of r_ves
        gvesopt=chpot              !standart energy
        alfavesopt=alfaves
        alfaoopt=alfaox
        alfaiopt=alfaix
        Rvesopt=r_ves
        tauoopt=tauo
        tauiopt=taui
        nvesopt=nves
        !write(*,*)'ch,ao,ai',Rvesopt,chpot,alfao,alfai
endif
if (g_v.LT.fmin) then       !(g_v.LT.fmin) then !(chpot.LT.chpotmin) then
!optimal for this value of r_ves
        fmin=g_v
        chpotmin=chpot
        alfaomin=alfaox
        alfaimin=alfaix
        tauomin=tauo
        tauimin=taui
```

```
!           alfa=alfaves
            !write(*,*)'!1=',alfaox,alfaix,fmin,r_ves

            endif
            enddo
            enddo
            chpot=chpotmin
            alfao=alfaomin
            alfai=alfaimin
!           alfaves=alfa
            tauo=tauomin
            taui=tauimin
            alfaox=alfaomin
            alfaix=alfaimin

!!          endif
!end do
!write(*,*)'ch,ao,ai',chpot,alfao,alfai
nvesi=agg_vesi(r_ves,taui,alfaix)
nveso=agg_veso(r_ves,tauo,taui,alfaox)
nves=agg_ves(r_ves,tauo,taui,alfaox,alfaix)
alfaves = (alfaox*nveso+ alfaix*nvesi)/nves
g_v=chpot-alfaves*dlog(X1A)-(1.0d0-alfaves)*dlog(X1B)

nn=nves
Xit = (X1A**(alfaves))*(X1B**(1.0d0 - alfaves))*dexp(- chpot)
if (Xit.lt.1.0d0)then
!Xg=Xit**nn
Xg=dlog(nn)+nn*dlog(Xit)
Xg=dexp(Xg)
!write(*,*)'!2',Xg
else
Xg=1.0d0
endif

write(31,'((1X,F8.3),4(2X,E12.6),4(5X,F9.5))')r_ves/ls_min,nn,Xit**nn,Xg,Xg*nn,chpot,alfaves,alfaox,alfaix

enddo
a_vi=a_vesi(rvesopt,tauiopt,alfaiopt)
a_vo=a_veso(rvesopt,tauoopt,tauiopt,alfaoopt)
agg_vo=agg_veso(rvesopt,tauoopt,tauiopt,alfaoopt)
agg_vi=agg_vesi(rvesopt,tauiopt,alfaiopt)
def =(agg_vo*g_def(tauoopt,0.0d0,alfaoopt,4.0d0) +agg_vi*g_def(tauiopt,0.0d0,alfaiopt,4.0d0))/nvesopt
ster= (agg_vo*StericFrEn(a_vo,alfaoopt)+agg_vi*StericFrEn(a_vi,alfaiopt))/nvesopt
```

```
tr=(agg_vi*TransferFrEn(alfaiopt)+agg_vo*TransferFrEn(alfaoopt))/nvesopt
mix=(agg_vi*DumMixFrEn(alfaiopt)+agg_vo*DumMixFrEn(alfaoopt))/nvesopt
int=(agg_vo*InterfacialFrEn(a_vo,alfaoopt)+agg_vi*InterfacialFrEn(a_vi,alfaiopt))/nvesopt
ion=g_ionves(rvesopt,tauoopt,tauiopt,alfaoopt,alfaiopt)
ion_nlin=g_ionves_nlin(rvesopt,tauoopt,tauiopt,alfaoopt,alfaiopt)
!  DifStandChPot = StericFrEn(a,alfagA) + DefFrEn(Rc,P,alfagA) + TransferFrEn(alfagA) +&                    !
free energy of nonionic micelles with deformation part as in Nagarajan model
!  InterfacialFrEn(a,alfagA) + MixFrEn(alfagA)
!
! ADD   DIPOLAR   PART  of FREE ENERGY
!
!if (MconstA(9).GT.0.or.MconstB(9).GT.0.) then                                    ! if at least one surfactant is
zwitterionic
 ! DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rc,0.0d0,1.0d0)
!elseif (MconstA(8)*MconstB(8)==2.)then                                                    !  cationic  +
anionic    OR   cationic+anionic
 !  DifstandChPot = DifstandChPot + g_zwitter(alfagA,Rc,0.0d0,1.0d0)                  !(alfa,b,c,sh)
!endif

if (ldamp.GT.0) write (18,'(2X,F8.4,13(2X,f12.5))') alfavesopt,def,tr,mix,&
int,ster,0.0d0,ion, ion_nlin,gvesopt,rvesopt,rvesopt+tauoopt+tauiopt,alfaoopt,alfaiopt
return
end
!*********************************************************************
real*8 FUNCTION StericFrEn(a, alfagA)
IMPLICIT none
REAL*8 a,alfagA
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
     nzero,krDeb,dchpotCH2,dchpotCH3,otn,Cadd,CsurA,CsurB
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
otn = (alfagA*MconstA(5) + (1. - alfagA)*MconstB(5)) / a
!write(*,*)'a=',a,'alfa=',alfagA
if (otn<1.) then
  StericFrEn = -DLOG(1.d0 - (alfagA*MconstA(5) + (1.d0 - alfagA)*MconstB(5)) / a)
else
  StericFrEn = 1000.
end if
END FUNCTION
!*********************************************************************
real*8 FUNCTION DefFrEn(Rs, P, alfagA)
IMPLICIT none
```

```
REAL*8, PARAMETER :: L = 4.6
real*8 NsegmA,NsegmB, Rs, P, alfagA,Bg,Qg,const
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
real*8 tip ! tip = 1 - вычисления для сфер, 2 - вычисления для цилиндров
common /tip/ tip
const = 9.d0
if(tip==2.) const = 10.d0
NsegmA = MconstA(3) / L
NsegmB = MconstB(3) / L
Bg = (const*P*Pi**2.)/80.d0
If (Rs < MconstB(3)) then
 Qg = Rs
else
 Qg = MconstB(3)
end if
DefFrEn = Bg*( (alfagA/NsegmA)*(Rs/L)**2. + ((1.0d0 - alfagA)/NsegmB)*(Qg/L)**2. )

END FUNCTION
!*********************************************************************
real*8 FUNCTION TransferFrEn(alfagA)
IMPLICIT none
Real*8 dchA,dchB,alfagA
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
 COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
dchA = ( MconstA(1)- 1.d0)*dchpotCH2 + dchpotCH3
dchB = ( MconstB(1)- 1.d0)*dchpotCH2 + dchpotCH3
TransferFrEn = alfagA*dchA + (1.d0 - alfagA)*dchB
END FUNCTION
!*********************************************************************
real*8 FUNCTION InterfacialFrEn(a, alfagA)
IMPLICIT none
REAL*8 a, alfagA,ettaA,sigmaagg,Cadd,CsurA,CsurB
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
```

```
      REAL*8
T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
       COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
          nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
      ettaA = alfagA*MconstA(2)/(alfagA*MconstA(2) + (1. - alfagA)*MconstB(2))
      sigmaagg = ettaA*sigmasA + (1. - ettaA)*sigmasB
      InterfacialFrEn = ( sigmaagg/(k*T) ) * (a - alfagA*MconstA(4) - (1.d0 - alfagA)*MconstB(4))*1.0d-16
      END FUNCTION
      !********************************************************************** real*8 function
DumMixFrEn(alfagA)                                    ! Regular mixing part, from Nagu and
solubility parameters      (mixing energy always positive)    ! Toggle          MixFrEn and DumMixFrEn
      implicit none
      real*8 ettaA,deltaHmix,alfagA,Cadd,CsurA,CsurB
      integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
      REAL*8
T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
       COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
          nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
      ettaA = alfagA*MconstA(2)/(alfagA*MconstA(2) + (1. - alfagA)*MconstB(2))
      if (ettaA==0.or.ettaA==1.) then
       DumMixFrEn = 0.
                                                    ! Toggle          MixFrEn and DumMixFrEn
      return
      end if
      deltaHmix = ettaA*MconstA(7) + (1.d0 - ettaA)*MconstB(7)
            ! Hildebrand solubility parameter
      DumMixFrEn = alfagA * dlog(ettaA) + (1.d0 - alfagA) * dlog(1.d0 - ettaA) + &
      ! Toggle          MixFrEn and DumMixFrEn
      (alfagA*MconstA(2)*(MconstA(7) - deltaHmix)**2 + &
      (1.d0 - alfagA)*MconstB(2)*(MconstB(7) - deltaHmix)**2)/k/T*1.d-17
      end function
      !**********************************************************************
      real*8 function MixFrEn(alfagA)                                    ! Flory mixing part,
mixing energy may be negative      ! Toggle          MixFrEn and DumMixFrEn
      implicit none
      real*8 ettaA,deltaHmix,alfagA,Cadd,CsurA,CsurB, Flochi, vs
      integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
      REAL*8
T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
       COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
          nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
      ettaA = alfagA*MconstA(2)/(alfagA*MconstA(2) + (1.d0 - alfagA)*MconstB(2))
      if (ettaA<1.0d-15.or.dabs(ettaA-1.0d0)<1.0d-15) then
       MixFrEn = 0.
                                    ! Toggle          MixFrEn and DumMixFrEn
```

```
        return
        end if
        Flochi=1.0d0/97.3d0                                    ! Flory parameter divided by Kuhn L^3 temperature dependence should be taken into account explicitly
        MixFrEn = alfagA * dlog(ettaA) + (1.d0 - alfagA) * dlog(1.d0 - ettaA) + &      ! Toggle MixFrEn and DumMixFrEn
        Flochi*alfagA*(1.d0 - alfagA)*MconstA(2)*MconstB(2)/vs(alfagA)
        end function
        !*************************************************************
            REAL*8 function g_tor(b,c,alfa)
            IMPLICIT none
                integer Ldamp
            REAL*8 b, c ,alfa, a_t, g_ion,g_ion_nlin,g_zwitter,www
            REAL*8 a_tor,InterfacialFrEn,TransferFrEn,g_def,StericFrEn,MixFrEn
                common /out/ ldamp
            a_t = a_tor(b,c,alfa)
            g_tor = InterfacialFrEn(a_t,alfa)
                g_tor=g_tor+g_def(b,c,alfa,3.0d0)
                g_tor=g_tor+TransferFrEn(alfa)
                g_tor=g_tor+StericFrEn(a_t,alfa)
                g_tor=g_tor+MixFrEn(alfa)
            www=g_ion(b,c,alfa,3.0d0)
            g_tor = g_tor+g_ion(b,c,alfa,3.0d0)
        !     g_tor = g_tor+g_ion_nlin(alfa,b,c,3.0d0)          ! calls ionic and g_zwitter(alfa,b,c,sh) via gXgsub - routines even if  surfactants are nonionic - non-zwitterionic
            if (ldamp.GT.0) write (16,'(10X,F8.4,10(2X,E12.5))') alfa,&
            g_def(b,c,alfa,3.0d0),TransferFrEn(alfa),MixFrEn(alfa), &
                 InterfacialFrEn(a_t,alfa),StericFrEn(a_t,alfa), &
            g_zwitter(alfa,b,c,3.0d0),g_ion(b,c,alfa,3.0d0),&
            g_ion_nlin(alfa,b,c,3.0d0),g_tor,b
             RETURN
             END
        !C*****************************************************************
            REAL*8 function g_pl(r_pl,alfa)
            IMPLICIT none
                integer Ldamp
            REAL*8 r_pl,alfa, a_p , g_ion,g_ion_nlin,g_zwitter
            REAL*8 a_pl, InterfacialFrEn,TransferFrEn,g_def,StericFrEn,MixFrEn
                common /out/ ldamp

                a_p=a_pl(r_pl,alfa)
                g_pl = InterfacialFrEn(a_p,alfa)
                g_pl=g_pl+g_def(r_pl,0.d0,alfa,0.d0)
                g_pl=g_pl+TransferFrEn(alfa)
```

```
                g_pl=g_pl+StericFrEn(a_p,alfa)
                g_pl=g_pl+MixFrEn(alfa)
            g_pl = g_pl+g_ion(r_pl,0.d0,alfa,0.d0)
        !     g_pl = g_pl+g_ion_nlin(alfa,r_pl,0.d0,0.d0)                    ! calls ionic and
g_zwitter(alfa,b,c,sh) via gXgsub - routines even if  surfactants are nonionic - non-zwitterionic
            if (ldamp.GT.0) then
                write (16,'(10X,F8.4,10(2X,E12.5))') alfa, &
            g_def(r_pl,0.d0,alfa,0.d0),TransferFrEn(alfa),MixFrEn(alfa),&
                 InterfacialFrEn(a_p,alfa), StericFrEn(a_p,alfa),&
            g_zwitter(alfa,r_pl,0.0d0,0.0d0),g_ion(r_pl,0.d0,alfa,0.d0),&
            g_ion_nlin(alfa,r_pl,0.d0,0.d0),g_pl,r_pl                !,v_patch(r_patch,c_jun))/vs(alfa_jun)
                endif
            RETURN
            END
        !C*********************************************************************
            REAL*8 function g_cyl(r_cyl,alfa)
            IMPLICIT none
            REAL*8 r_cyl,alfa, a_c , g_ion,g_ion_nlin
                REAL*8 a_cyl,InterfacialFrEn,TransferFrEn,g_def,StericFrEn,MixFrEn

                a_c=a_cyl(r_cyl,alfa)
            g_cyl = InterfacialFrEn(a_c,alfa)
                g_cyl=g_cyl+g_def(r_cyl,0.d0,alfa,1.0d0)
                g_cyl=g_cyl+TransferFrEn(alfa)
                g_cyl=g_cyl+StericFrEn(a_c,alfa)
                g_cyl=g_cyl+MixFrEn(alfa)
         !   g_cyl = g_cyl+g_ion(r_cyl,0.d0,alfa,1.0d0)
            g_cyl = g_cyl+g_ion_nlin(alfa,r_cyl,0.d0,1.0d0)                ! (alfa,b,c,3.0d0)calls ionic and
g_zwitter(alfa,b,c,sh) via gXgsub - routines even if  surfactants are nonionin - non-zawitterionic
            RETURN
            END
        !C*********************************************************************
            REAL*8 function g_sph(r_sph,alfa)
            IMPLICIT none
            REAL*8 r_sph,alfa, a_s, g_ion,g_ion_nlin
                REAL*8 a_sph,InterfacialFrEn,TransferFrEn,g_def,StericFrEn,MixFrEn
                a_s=a_sph(r_sph,alfa)
                g_sph = InterfacialFrEn(a_s,alfa)
                g_sph=g_sph+g_def(r_sph,0.d0,alfa,2.0d0)
                g_sph=g_sph+TransferFrEn(alfa)
                g_sph=g_sph+StericFrEn(a_s,alfa)
                g_sph=g_sph+MixFrEn(alfa)
         !   g_sph = g_sph+g_ion(r_sph,0.d0,alfa,2.0d0)
```

```
      g_sph = g_sph+g_ion_nlin(alfa,r_sph,0.d0,2.0d0)                    ! calls ionic
and g_zwitter(alfa,b,c,sh) via gXgsub - routines even if surfactants are nonionin - non-zawitterionic
      RETURN
      END
!C*****************************************************************
      REAL*8 function g_jun(b,c,alfa)                    !works when depend=1;  if alfa torus and alfa
plane are same, set alpl=alfa, if different, set alpl=alfa_pl (optimal the for plane)
      IMPLICIT none
      REAL*8 b, c, alfa,alfa_pl,alpl
          COMMON/optpl/r_patch,alfa_pl,g_patch
          REAL*8 r_patch, r_opt_pl
      REAL*8 denomT,denomP,psi,vs
      REAL*8 eta_patch, eta_torus, g_patch, g_torus
      REAL*8 v_tor_b,v_patch,g_pl,g_tor
      if(1)  then
          if( b < 0.0) then
           write(*,*) "error in g_jun(): b < 0"
           stop
          else
          if( c < 0.0) then
           write(*,*) "error in g_jun(): c < 0"
           stop
          endif
          endif
          endif
         r_opt_pl=r_patch
         alpl =alfa_pl
!C       alpl =alfa
      psi = vs(alpl)/vs(alfa)
      denomT = v_tor_b(b,c)+v_patch(r_opt_pl,c)/psi
      denomP = v_tor_b(b,c)*psi+v_patch(r_opt_pl,c)
      eta_torus = v_tor_b(b,c)/denomT
      eta_patch = v_patch(r_opt_pl,c)/denomP
      g_patch = g_pl(r_opt_pl,alpl)
      g_torus = g_tor(b,c,alfa)
      g_jun = g_patch*eta_patch+g_torus*eta_torus
       RETURN
       END
!C*****************************************************************
      REAL*8 function g_jun2(b,c,alfa)                   !works when depend=0;  if alfa torus and alfa
plane are same, set alpl=alfa, if different, set alpl=alfa_pl (optimal the for plane)
      IMPLICIT none
      REAL*8 b, c,alfa,r_patch,alfa_pl, alpl
          COMMON/optpl/r_patch,alfa_pl,g_patch
```

```
      REAL*8 r_opt_pl
      Common/nprint/nprint
      Integer nprint
      REAL*8 denomT,denomP,psi,vs
    REAL*8 eta_patch, eta_torus, g_patch, g_torus
    REAL*8 v_tor_b,v_patch,g_pl,g_tor
    if(1)  then
      if( b < 0.0) then
        write(*,*) "error in g_jun2(): b < 0"
        stop
      else
      if( c < 0.0) then
        write(*,*) "error in g_jun2(): c < 0"
        stop
      endif
      endif
      endif
      r_opt_pl=r_patch
      alpl =alfa_pl
    psi = vs(alpl)/vs(alfa)
    denomT = v_tor_b(b,c)+v_patch(r_opt_pl,c)/psi
    denomP = v_tor_b(b,c)*psi+v_patch(r_opt_pl,c)
    eta_torus = v_tor_b(b,c)/denomT
    eta_patch = v_patch(r_opt_pl,c)/denomP
    g_patch = g_pl(r_opt_pl,alpl)
    g_torus = g_tor(b,c,alfa)
    g_jun2 = g_patch*eta_patch+g_torus*eta_torus
    RETURN
    END
!**********************************************************************
    REAL*8 function g_def(b,c,alfa,sh)
    IMPLICIT none
    REAL*8 b,c,alfa, ratio, res, sh
      REAL*8, PARAMETER :: L = 4.6
    real*8 NsegmA,NsegmB
    integer, parameter :: ch = 9
    real*8 MconstA(ch), MconstB(ch)
    REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
    real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
    COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
     epsilon,MconstA, MconstB,&
      nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB

    !pi=3.141592653589793
```

```
                NsegmA = MconstA(3) / L
                    NsegmB = MconstB(3) / L

            ratio=pi**2*(alfa/NsegmA + (1.0d0- alfa)/NsegmB)*(b/L)**2
                IF (sh.eq.4) then
              res = 10.0d0/80.d0*ratio
            ELSE
                IF (sh.eq.3) then
              res = 3.0d0/8.d0*(pi*c/12-b/15)/(pi*c/2-2.d0/3.d0*b)*ratio                    ! correct
            ELSE
            IF (sh.eq.2) then
              res = 3.0d0/80.d0*ratio
            ELSE
            IF (sh.eq.1) then
              res = 5.0d0/80.d0*ratio
            ELSE
            IF (sh.eq.0) then
              res = 10.0d0/80.d0*ratio
            ELSE
              PRINT("wrong shape parameter 'sh' in g_def(...)")
            ENDIF
            ENDIF
            ENDIF
            ENDIF
                  ENDIF
            g_def = res
            RETURN
            END
          !*********************************************************************
             REAL*8 function g_vesout(b,c,d,alfao)                                !b=r_ves,c=tauo  thickness  of
outer layer
            IMPLICIT none
                !d=taui
            REAL*8 b, c,d
            REAL*8 StericFrEn, g_def
            REAL*8 DumMixFrEn,alfao,TransferFrEn,InterfacialFrEn
            REAL*8 a_vo, a_veso
            a_vo=a_veso(b,c,d,alfao)
            g_vesout =g_def(c,0.0d0,alfao,4.0d0) +StericFrEn(a_vo,alfao)
            g_vesout=g_vesout+TransferFrEn(alfao)+DumMixFrEn(alfao)
                g_vesout=g_vesout+InterfacialFrEn(a_vo,alfao)
            RETURN
            END
```

```
!C**********************************************************************
      REAL*8 function g_vesin(b,c,alfai)                    !b=r_ves,c=taui thickness of inner layer
      IMPLICIT none
      REAL*8 b, c
      REAL*8 StericFrEn, g_def
      REAL*8 DumMixFrEn,alfai,TransferFrEn,InterfacialFrEn
      REAL*8 a_vi, a_vesi
      a_vi=a_vesi(b,c,alfai)

        g_vesin =g_def(c,0.0d0,alfai,4.0d0)+StericFrEn(a_vi,alfai)
      g_vesin=g_vesin+TransferFrEn(alfai)+DumMixFrEn(alfai)
        g_vesin=g_vesin+InterfacialFrEn(a_vi,alfai)
      RETURN
      END
!C**********************************************************************
      REAL*8 function g_ves(b,c,d,alfao,alfai)
      IMPLICIT none
      REAL*8 b, c, d
!b=r_ves,c=tauo,d=taui
      REAL*8 g_ionves,g_ionves_nlin,g_vesin,g_vesout
      REAL*8 alfao,alfai
      REAL*8 a_ves,a_v,g_v,alfa
      REAL*8 agg_veso, agg_vesi,agg_ves,agg_v,agg_vo,agg_vi

      agg_vo=agg_veso(b,c,d,alfao)
      agg_vi=agg_vesi(b,d,alfai)
      agg_v=agg_ves(b,c,d,alfao,alfai)
        g_ves=(agg_vi*g_vesin(b,d,alfai)+agg_vo*g_vesout(b,c,d,alfao))/agg_v
      g_ves=g_ves+g_ionves(b,c,d,alfao,alfai)
!       g_ves=g_ves+g_ionves_nlin(b,c,d,alfao,alfai)
      RETURN
      END
!C**********************************************************************
      real*8 FUNCTION volume(Nc, T)
      IMPLICIT none
      REAL*8 T,volCH3,volCH2, Nc
      volCH3 = 54.6 + 0.124*(T - 298.)
      volCH2 = 26.9 + 0.0146*(T - 298.)
      volume = volCH3 + (Nc - 1.)*volCH2
      END FUNCTION
      !**********************************************************************
      real*8 FUNCTION length(Nc)
      IMPLICIT none
```

```
REAL*8 Nc
length = 1.50d0 + 1.265d0*Nc
END FUNCTION
!*****************************************************************
    REAL*8 function a_tor_b(b,c)
!C Total area of semi-torus part
    IMPLICIT none
    REAL*8 b, c, Pi
    pi=3.141592653589793
    a_tor_b = Pi*b*(Pi*c-2*b)
    RETURN
    END
!C*****************************************************************
    REAL*8 function v_tor_b(b,c)
!C Total volume of semi-torus part
    IMPLICIT none
    REAL*8 b,c,Pi
    pi=3.141592653589793
    v_tor_b = pi*b**2*(pi*c/2-2.d0/3.d0*b)
    RETURN
    END
!C*****************************************************************
    REAL*8 function a_patch(b,c)
!C Total area of bilayer patch
    IMPLICIT none
    REAL*8 b,c
    REAL*8 pi
    pi=3.141592653589793
    a_patch = 2*c**2*(dsqrt(3.d0)-pi/2)
    b=b
    RETURN
    END
!C*****************************************************************
    REAL*8 function v_patch(b,c)
!C Total volume of bilayer patch
    IMPLICIT none
    REAL*8 b,c
    REAL*8 pi
    pi=3.141592653589793
    v_patch = 2*b*c**2*(dsqrt(3.d0)-pi/2)
    RETURN
    END
!C*****************************************************************
    REAL*8 function a_sph(r_sph,alfa)
```

```
!C Area per surfactant molecule for sphere
      IMPLICIT none
      REAL*8 r_sph,alfa, vs
            external vs
      a_sph = 3*vs(alfa)/r_sph
      RETURN
      END
!C*********************************************************************
      REAL*8 function vs(alfa)
            implicit none
            REAL*8 alfa
            integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
!C                                  1     2     3     4     5     6     7     8      9
!C                                  Nc    Vs    ls    a0    ap   delta deltaH  logic   Ddip
      REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
      real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
            real*8 Cadd,CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
     epsilon, MconstA, MconstB, nzero,krDeb,dchpotCH2,dchpotCH3,&
             Cadd,CsurA,CsurB

            vs = alfa*MconstA(2)+(1.0d0-alfa)*MconstB(2)
            RETURN
            END
!C*********************************************************************
            REAL*8 function eta1(alfa)
            implicit none
            REAL*8 vs,alfa
            integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
!C                     !  1     2     3     4     5     6     7     8      9
!C                     !  Nc    Vs    ls    a0    ap   delta deltaH  logic   ddip
      REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
      real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
            real*8 Cadd,CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
     epsilon, MconstA, MconstB, nzero,krDeb,dchpotCH2,dchpotCH3,&
             Cadd,CsurA,CsurB
            eta1=alfa*MconstA(2)/vs(alfa)
            RETURN
            END
!C*********************************************************************
            REAL*8 function eta2(alfa)
```

```
      implicit none
      REAL*8 vs,alfa
      integer, parameter :: ch = 9
    real*8 MconstA(ch), MconstB(ch)
!C                  !  1    2    3    4    5    6    7    8    9
!C                  !  Nc   Vs   ls   a0   ap   delta deltaH  logic    ddip
    REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
    real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
      real*8 Cadd,CsurA,CsurB
    COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
   epsilon, MconstA, MconstB, nzero,krDeb,dchpotCH2,dchpotCH3,&
      Cadd,CsurA,CsurB
      eta2=(1.0d0-alfa)*MconstB(2)/vs(alfa)
      RETURN
      END
!C*********************************************************************
      REAL*8 function RUbound(alfa)
      implicit none
      REAL*8 alfa, eta1,eta2
      integer, parameter :: ch = 9
    real*8 MconstA(ch), MconstB(ch)
!C                  !  1    2    3    4    5    6    7    8    9
!C                  !  Nc   Vs   ls   a0   ap   delta deltaH  logic    ddip
    REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
    real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3
      real*8 Cadd,CsurA,CsurB
    COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
   epsilon, MconstA, MconstB, nzero,krDeb,dchpotCH2,dchpotCH3, &
      Cadd,CsurA,CsurB
      RUbound=eta1(alfa)*MconstA(3)+eta2(alfa)*MconstB(3)
      RETURN
      END
!C*********************************************************************
    REAL*8 function a_cyl(r_cyl,alfa)
    ! Area per surfactant molecule for cylinder
    IMPLICIT none
    REAL*8 r_cyl,vs,alfa
    a_cyl = 2*vs(alfa)/r_cyl
    RETURN
    END
!C*********************************************************************
    REAL*8 function a_tor(b,c,alfa)
!C Average area per surfactant molecule in semi-torus part
    IMPLICIT none
```

```
      REAL*8 b,c, alfa, vs
      REAL*8 a_tor_b,v_tor_b
      a_tor = vs(alfa)*a_tor_b(b,c)/v_tor_b(b,c)
      RETURN
      END
!C*****************************************************************
      REAL*8 function a_pl(r_pl,alfa)
!C Area per surfactant molecule for plane
      IMPLICIT none
      REAL*8 r_pl, alfa, vs
      a_pl = vs(alfa)/r_pl
      RETURN
      END
!C*****************************************************************
      real*8 function adelta(sh,alfa,b,c,delta)                    ! surface area at distance b+delta from
center
            implicit none
            real*8 sh,alfa,b,c,delta,vs,a_tor_b,v_tor_b
            if (sh.eq.5)then
                  !vesicle,inner layer
            adelta=3*vs(alfa)*(b+delta)**2/((b+c)**3-b**3)
!r_ves=b,taui=c,delta<0
             endif
      if (sh.eq.4)then
                  !vesicle,outer layer
            adelta=3*vs(alfa)*(b+c+delta)**2/((b+c)**3-b**3)
!tauo=c,r_ves+taui=b
       endif
      if (sh.eq.3)then
            adelta=a_tor_b((b+delta),c)*vs(alfa)/v_tor_b(b,c)
            return
            endif
      if (sh.eq.2)then
            adelta=3*vs(alfa)*(b+delta)**2/b**3
      else
            if (sh.eq.1)then
            adelta=2*vs(alfa)*(b+delta)/b**2
      endif
            endif
            IF (sh.eq.0)then
            adelta=vs(alfa)/b
            endif
      RETURN
      END
!C*****************************************************************
```

```
real*8 function  Jun_alf(b,c,r_patch,alfa_pl,alfa_jun)
            implicit none
            real*8 b,c,r_patch,alfa_pl,alfa_jun,vs,v_tor_b,v_patch,n_tor,n_pl,njun

n_tor =    v_tor_b(b,c)/vs(alfa_jun)
n_pl =    v_patch(r_patch,c)/vs(alfa_pl)
njun = n_tor+n_pl
Jun_alf = (n_tor*alfa_jun + n_pl*alfa_pl)/njun

return
end
!C*****************************************************************
subroutine  Av_alf_jun(b,c,r_patch,alfa_pl,alfa_jun,x,n_tor,n_pl,njun)
            implicit none
            real*8 b,c,r_patch,alfa_pl,alfa_jun,vs,v_tor_b,v_patch,n_tor,n_pl,njun,x

n_tor =    v_tor_b(b,c)/vs(alfa_jun)
n_pl =    v_patch(r_patch,c)/vs(alfa_pl)
njun = n_tor+n_pl
x = (n_tor*alfa_jun + n_pl*alfa_pl)/njun                    !  average fraction of surfactant A in junction
return
end
!C*********************************************************************
      REAL*8 function a_ves(r_ves,tauo,taui,alfao,alfai)
!C Area per surfactant molecule for vesicle
     IMPLICIT none
     REAL*8 r_ves, agg_ves,agg_v, tauo,taui,pi,alfao,alfai
            pi=3.141592653589793
            agg_v=agg_ves(r_ves,tauo,taui,alfao,alfai)
     a_ves = 4*pi*((r_ves+tauo+taui)**2+r_ves**2)/agg_v
     RETURN
     END
!C*********************************************************************
     REAL*8 function a_veso(r_ves,tauo,taui,alfao)
!C Area per surfactant molecule in outer layer for vesicle
     IMPLICIT none
     REAL*8 r_ves, agg_veso,agg_vo,tauo,taui,pi,alfao
            pi=3.141592653589793
            agg_vo=agg_veso(r_ves,tauo,taui,alfao)
     a_veso = 4*pi*(r_ves+tauo+taui)**2/agg_vo
     RETURN
     END
!C*********************************************************************
     REAL*8 function a_vesi(r_ves,taui,alfai)
```

```
!C Area per surfactant molecule in inner layer for vesicle
      IMPLICIT none
      REAL*8 r_ves,taui,alfai,agg_vesi,pi
            pi=3.141592653589793
      a_vesi = 4*pi*r_ves**2/agg_vesi(r_ves,taui,alfai)
      RETURN
      END
!C*********************************************************************
      REAL*8 function agg_veso(r_ves,tauo,taui,alfao)
      IMPLICIT none
      REAL*8 r_ves,tauo,taui,pi,alfao, vs
            pi=3.141592653589793
      agg_veso=4*pi*((r_ves+tauo+taui)**3-(r_ves+taui)**3)/(3*vs(alfao))
            !aggregation number of outer layer
      RETURN
      END
!C*********************************************************************
      REAL*8 function agg_vesi(r_ves,taui,alfai)
      IMPLICIT none
      REAL*8 r_ves,taui,pi,alfai,vs
      pi=3.141592653589793
            agg_vesi=4*pi*((r_ves+taui)**3-r_ves**3)/(3*vs(alfai))
            !aggregation number of inner layer
      RETURN
      END
!C*********************************************************************
      REAL*8 function agg_ves(r_ves,tauo,taui,alfao,alfai)
      IMPLICIT none
      REAL*8 r_ves,tauo,taui,agg_vesi, agg_veso, alfao,alfai
      agg_ves=agg_veso(r_ves,tauo,taui,alfao)+agg_vesi(r_ves,taui,alfai)
             !total aggregation number
      RETURN
      END
!C*********************************************************************
Subroutine alf_ion_Deb(alfagA,alfagion,delta)
!  calculates delta, alfagion, krDeb  from  micellar composition   -   alagA
implicit none
REAL*8 Cadd,Cone,X1A,X1B,X1ionic,alfagA,alfaA1
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)   ! molecular data
!   1     2     3     4     5     6     7     8              9
!  Nc     Vs     ls     a0     ap    delta deltaH  logic  ddip
! logic =  1 - cationic surf, 2 - anionic surf, 3 - nonionic surf., when   ddip>0 zwitterionic surf.
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
```

```
     nzero,krDeb,dchpotCH2,dchpotCH3,CsurA,CsurB
COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
      nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
common /monomers/ X1A,X1B,alfaA1
real*8 alfagion, delta, Vmaqua

 If (MconstA(8)/=3.and.MconstB(8)/=3.) then
   X1ionic = X1A + X1B
 elseif (MconstA(8)==3.and.MconstB(8)/=3.) then
   X1ionic = X1B
 elseif (MconstB(8)==3.and.MconstA(8)/=3.) then
  X1ionic = X1A
 else
          X1ionic = 0.d0
 end if

if (MconstA(8)/=3.or.MconstB(8)/=3.) then
 Vmaqua = 18.0152/998.2
 Cone = X1ionic/Vmaqua
 nzero = (Cone + Cadd)*Nav/1000.
!  if(Cone<0.d0) stop                                                    ! C in [M],  nzero in [ions/cm^3]
 krDeb = (1.0d-8)*dsqrt(8.*Pi*nzero*e**2./ (epsilon * k * T))  !  [A^-1]      8*Pi*l_B*nzero    l_B  in [cm]
CGSE
end if

If (MconstA(8)==3.and.MconstB(8)==3.) then
 alfagion = 0.0d0
elseif (MconstA(8)==3.and.MconstB(8)/=3.) then
 delta = MconstB(6)
 alfagion = (1.0d0 - alfagA)
elseif (MconstB(8)==3.and.MconstA(8)/=3.) then
 delta = MconstA(6)
 alfagion = alfagA
elseif (MconstA(8)*MconstB(8)==1.or.MconstA(8)*MconstB(8)==4.) then
 alfagion = 1.0d0
 delta = alfagA*MconstA(6) + (1.0d0 - alfagA)*MconstB(6)
else
 delta = alfagA*MconstA(6) + (1.0d0 - alfagA)*MconstB(6)
 alfagion = dabs (2.0d0*alfagA - 1.0d0)
end if
return
end
!**********************************************************************
Subroutine alf_dip(alfa,alfag_dip,delta_dip,Ddip)
```

```
! calculates delta-dip, alfag_dip krDeb from micellar composition - alagA
implicit none
REAL*8 Cadd,X1A,X1B,alfaA1
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch) ! массив молекулярных данных
! 1 2 3 4 5 6 7 8 9
! Nc Vs ls a0 ap delta deltaH logic ddip
! logic = 1 - катионные ПАВ, 2 - анионные ПАВ, 3 - неионные ПАВ ddip>0 zwitterionic
REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
   nzero,krDeb,dchpotCH2,dchpotCH3,CsurA,CsurB
COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
    nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
common /monomers/ X1A,X1B,alfaA1
real*8 alfag_dip, alfa, Ddip, delta_dip

alfag_dip = 0.0d0
If (MconstA(8)*MconstB(8)==2.) then                                    ! anionic+ cationic OR cationic +anionic
 alfag_dip = min(alfa,(1.0d0-alfa))                                    ! possibly, coefficient 2 is needed
 delta_dip = min(MconstA(6),MconstB(6))
 Ddip = dabs(MconstA(6)-MconstB(6))
elseif ( (MconstA(9).GT.0).and.(MconstB(9).LT.0) ) then                ! A-zwitterionic, B -nonionic or ionic
 alfag_dip = alfa
 delta_dip = MconstA(6)
 Ddip = MconstA(9)
elseif ( (MconstB(9).GT.0).and.(MconstA(9).LT.0) ) then                ! B-zwitterionic, A -nonionic or ionic
 alfag_dip = (1.0d0 - alfa)
 delta_dip = MconstB(6)
 Ddip = MconstB(9)
elseif ( (MconstA(9).GT.0).and.(MconstB(9).GT.0) ) then                ! both surfactants are zwitterionic with same head group
 alfag_dip = 1.0d0
 delta_dip = alfa*MconstA(6) + (1.0d0 - alfa)*MconstB(6)
 Ddip = alfa*MconstA(9) + (1.0d0 - alfa)*MconstB(9)                    ! Questionable, when dipole groups are different
else
 write (*,*) 'NO DIPOLE MOMENTS !!!!!!'
 Ddip = 0.0d0
end if
return
end
!**********************************************************************
   REAL*8 function g_ion(b,c,alfa,sh)                                  ! linear PB
   IMPLICIT none
```

```
      REAL*8 b, c, sh, alfa, achrg
      REAL*8 m, p, s, theta, x0, alfagion, delta, adelta
      REAL*8 y_sph, y_tor, y_cyl, y_pl, l_b
      integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
!C          !  1    2    3    4    5    6    7    8    9
!C          !  Nc   Vs   ls   a0   ap   delta deltaH  logic  ddip
      REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
      epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
        CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
      epsilon,MconstA, MconstB,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
      CsurA,CsurB

      call alf_ion_Deb(alfa,alfagion,delta)          ! input alfa - aggregate composition at current monomer composition; output  alfagion - charge fraction in the aggregate, delta -distance

      if (alfagion<1.0d-15)then                      ! no ionic surfactants in the system
      g_ion = 0.0d0
      return
      endif

      l_b = e**2/(epsilon*k*T)*1.0d8                 ! [A]   from  CGSE units /cm/
      achrg=adelta(sh,alfa,b,c,delta)                ! area per molecule at b+delta
      s =4*Pi*l_b*alfagion/(achrg*krDeb)
      x0 = krDeb*(b + delta)

      IF (sh.eq.3) then
       theta = 3.d0/4.d0*pi
       m = (c+2*b*cos(theta))/(c+b*cos(theta))
       p = (m-1.d0)/2.d0
!      if ((m-1.0d0).LT.3.0d-3) p=0
       g_ion = s*achrg*krDeb*y_tor(p,x0,x0,s)/(8*Pi*l_b)

!      print*,'linear',g_ion,m,s,achrg,krDeb,l_b,x0
      ELSE
      IF (sh.eq.2) then
       g_ion = s*achrg*krDeb*y_sph(x0,x0,s)/(8*Pi*l_b)
      ELSE
      IF (sh.eq.1) then

       g_ion = s*achrg*krDeb*y_cyl(x0,x0,s)/(8*Pi*l_b)
      ELSE
      IF (sh.eq.0) then
```

```
            g_ion = s*achrg*krDeb*y_pl(x0,x0,s)/(8*Pi*l_b)
          ELSE
              write(*,*) "Error in g_ion()"
          ENDIF
          ENDIF
          ENDIF
          ENDIF
          RETURN
          END
      !C*****************************************************************
              REAL*8 function g_ionves(b,c,d,alfao,alfai)                    ! linear PB
          IMPLICIT none
              REAL*8 b, c, d, alfai, alfao, achrg_in, achrg_out,l_b,g_in,g_out
          REAL*8  s_in, s_out,  x0,x00, alfagion, delta, adelta
          REAL*8 y_vi, y_vo,y_veso, y_vesi
              real*8 agg_ves,agg_vesi,agg_veso,agg_v,agg_vi,agg_vo
          integer, parameter :: ch = 9
          real*8 MconstA(ch), MconstB(ch)
      !C          !  1    2    3    4    5    6    7    8    9
      !C          !  Nc   Vs   ls   a0   ap   delta deltaH  logic  ddip
          REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
              epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
              CsurA,CsurB
          COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
              epsilon,MconstA, MconstB,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
              CsurA,CsurB
              l_b = e**2/(epsilon*k*T)*1.0d8                    ! [A]   from  CGSE units /cm/
              agg_vi=agg_vesi(b,d,alfai)
          agg_vo=agg_veso(b,c,d,alfao)
              agg_v=agg_ves(b,c,d,alfao,alfai)
              call alf_ion_Deb(alfao,alfagion,delta)          ! input alfa - aggregate composition at current
monomer composition; output  alfagion - charge fraction in the aggregate, delta -distance
              if (dabs(alfagion).le.1.0d-15)then                    ! no ionic surfactants in the
system
              alfagion = 0.0d0
              endif
              achrg_out=adelta(4.0d0,alfao,b+d,c,delta)             !     area per molecule at
b+c+d+delta   (outer layer)
              s_out =4*Pi*l_b*alfagion/(achrg_out*krDeb)


              call alf_ion_Deb(alfai,alfagion,delta)
              if (dabs(alfagion).le.1.0d-15)then                    ! no ionic surfactants in the
system
              alfagion = 0.0d0
              endif
```

```
            achrg_in=adelta(5.0d0,alfai,b,d,-delta)                    ! area per molecule at b-delta
      (inner layer)
            s_in =4*Pi*l_b*alfagion/(achrg_in*krDeb)

            x00=krDeb*(b + c + d+delta)
            x0 = krDeb*(b - delta)
            y_vi=y_vesi(x0,x0,s_in,x00,s_out)
            g_in=agg_vi*s_in*achrg_in*krDeb*y_vi
            y_vo=y_veso(x00,x0,s_in,x00,s_out)
            g_out=agg_vo*s_out*achrg_out*krDeb*y_vo

            g_ionves=(g_in+g_out)/(8*Pi*l_b*agg_v)
            ! write(*,*)'ion',g_ionves,alfai,y_vi,alfao,y_vo
            RETURN
          END
      !C*********************************************************************
            REAL*8 function g_ionves_nlin(b,c,d,alfao,alfai)
          IMPLICIT none
          REAL*8 b, c, d, alfai, alfao, achrg_in, achrg_out,l_b,g_in,g_out
          REAL*8  s_in, s_out, alfagion, delta, adelta,const
             real*8 agg_ves,agg_vesi,agg_veso,agg_v,agg_vi,agg_vo
          integer, parameter :: ch = 9
          real*8 MconstA(ch), MconstB(ch)
      !C            ! 1    2    3    4    5    6    7    8    9
      !C            ! Nc   Vs   ls   a0   ap  delta deltaH  logic  ddip
          REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
              epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
          COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
              epsilon,MconstA, MconstB,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
            l_b = e**2/(epsilon*k*T)*1.0d8                             ! [A]   from  CGSE units /cm/
            agg_vi=agg_vesi(b,d,alfai)
          agg_vo=agg_veso(b,c,d,alfao)
            agg_v=agg_ves(b,c,d,alfao,alfai)

            call alf_ion_Deb(alfao,alfagion,delta)          ! input alfa - aggregate composition at current
      monomer composition; output  alfagion - charge fraction in the aggregate, delta -distance

            if (dabs(alfagion).le.1.0d-15)then                         ! no ionic surfactants in the
      system
            alfagion = 0.0d0
            endif
          achrg_out=adelta(4.0d0,alfao,b+d,c,delta)                    !    area per molecule at
      b+c+d+delta     (outer layer)
            s_out =4*Pi*l_b*alfagion/(achrg_out*krDeb)
```

```
            call alf_ion_Deb(alfai,alfagion,delta)
            if (dabs(alfagion).le.1.0d-15)then                    ! no ionic surfactants in the
system
            alfagion = 0.0d0
            endif

            achrg_in=adelta(5.0d0,alfai,b,d,-delta)               ! area per molecule at b-delta
      (inner layer)
            s_in =4*Pi*l_b*alfagion/(achrg_in*krDeb)
                                                                  !b=r_ves, c+d=total thickness
            const=sqrt(1+(s_in/2)**2)
        if (dabs(s_in).le.1.0d-15)then                        ! no ionic surfactants in the system
            g_in = 0.0d0
            else
            g_in=agg_vi*2*(log(s_in/2+const)-2*(const-1)/s_in)
            g_in=g_in+ 4*log(0.5+0.5*const)/(krDeb*s_in*(b-delta))
            endif
            const=sqrt(1+(s_out/2)**2)
             if (dabs(s_out).le.1.0d-15)then                      ! no ionic surfactants in the
system
            g_out = 0.0d0
            else
          g_out=agg_vo*2*(log(s_out/2+const)-2*(const-1)/s_out)
            g_out=g_out-4*log(0.5+0.5*const)/(krDeb*s_out*(b+c+d+delta))
             end if
            g_ionves_nlin=(g_in+g_out)/agg_v
            RETURN
          END
      !C*********************************************************************
          REAL*8 function y_tor(p,x,x0,s)
          IMPLICIT none
          REAL*8 p, x, x0, s
          REAL*8 dimm1(1), dimm2(1)
          CALL dbsks(p,x,1,dimm1)
          CALL dbsks(p+1,x0,1,dimm2)
          y_tor = s*(x0/x)**p*dimm1(1)/dimm2(1)
          RETURN
          END
      !C*********************************************************************
          REAL*8 function y_pl(x,x0,s)
          IMPLICIT none
          REAL*8 x, x0, s
          y_pl = s*dexp(x0-x)
          RETURN
```

```
      END
!C*********************************************************************
      REAL*8 function y_cyl(x,x0,s)
      IMPLICIT none
      REAL*8 x, x0, s
      REAL*8 dbsk0, dbsk1
      y_cyl = s*dbsk0(x)/dbsk1(x0)
      RETURN
      END
!C*********************************************************************
      REAL*8 function y_sph(x,x0,s)
      IMPLICIT none
      REAL*8 x, x0, s
      y_sph = s*x0**2/(x0+1)*dexp(x0-x)/x
      RETURN
      END
!*********************************************************************
            REAL*8 function y_memb(x_in,s_in,x_out,s_out)        !for vesicles transmembrane potential
      IMPLICIT none
      REAL*8 x_out, x_in, s_in,s_out,const,epsilon_hc
            integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
      REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
      real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
            COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
            epsilon,MconstA, MconstB,nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
            const=(dcosh(x_in)/dsinh(x_in)-1/x_in)**(-1)
            epsilon_hc=2.1
       y_memb = s_in*const-s_out*x_out/(1+x_out)
            y_memb=y_memb/(1-epsilon_hc/(epsilon*(x_in-x_out))*(x_out*const/x_in+x_in/(x_out+1)))
            RETURN
            END
!C*********************************************************************
      REAL*8 function y_vesi(x,x_in,s_in,x_out,s_out)       !for vesicles potential of inner layer
      IMPLICIT none
      REAL*8 x, x_in,x_out, s_in,s_out,s_eff_in
      y_vesi = s_eff_in(x_in,s_in,x_out,s_out)*(x_in**2)*dsinh(x)/(x*(x_in*dcosh(x_in)-dsinh(x_in)))
            RETURN
            END
!C*********************************************************************
      REAL*8 function y_veso(x,x_in,s_in,x_out,s_out)      !for vesicles potential of outer layer
      IMPLICIT none
      REAL*8 x, x_out,x_in, s_out,s_in,s_eff_out
      y_veso = s_eff_out(x_in,s_in,x_out,s_out)*x_out**2/(x_out+1)*dexp(x_out-x)/x
```

```
      RETURN
          END
!C*********************************************************************
          REAL*8 function s_eff_in(x_in,s_in,x_out,s_out)        !for vesicles effective surf charge density of inner layer
          IMPLICIT none
      REAL*8 x_out,x_in, s_out,s_in,epsilon_hc,y_memb
          integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
      REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
      real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
           epsilon,MconstA, MconstB,nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB

          epsilon_hc=2.1
          s_eff_in=s_in+epsilon_hc*x_out*y_memb(x_in,s_in,x_out,s_out)/(x_in*epsilon*(x_in-x_out))
      RETURN
          END
!C*********************************************************************
          REAL*8 function s_eff_out(x_in,s_in,x_out,s_out)        !for vesicles effective surf charge density of outer layer
          IMPLICIT none
      REAL*8 x_out,x_in, s_out,s_in,epsilon_hc,y_memb
          integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
      REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
      real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
           epsilon,MconstA, MconstB,nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
          epsilon_hc=2.1
          s_eff_out=s_out-epsilon_hc*x_in*y_memb(x_in,s_in,x_out,s_out)/(x_out*epsilon*(x_in-x_out))
      RETURN
          END
!C*********************************************************************
      REAL*8 function g_ion_nlin(alfa,b,c,sh)
      IMPLICIT none
          REAL*8 b, c, sh, alfa
      REAL*8 m, s, theta, x0, alfagion, delta, adelta
      REAL*8 Evans1984,Evans1983,DUM_Ohshima1982
      integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
            !  1     2    3    4    5    6    7     8      9
            !  Nc    Vs   ls   a0   ap   delta deltaH  logic  ddip
      real*8 Ohshima_g_diff,achrg
```

```
      REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
      epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
            CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
     epsilon,MconstA, MconstB,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
     CsurA,CsurB

            call alf_ion_Deb(alfa,alfagion,delta)

            if (alfagion<1.0d-15) then          ! no ionic surfactants in the system
            g_ion_nlin = 0.0d0
            return
            endif
      achrg=adelta(sh,alfa,b,c,delta)                    ! surface per molecule at    b+delta
            s =4*Pi*alfagion*e**2/(epsilon*krDeb*achrg*(1.0d-8)*k*T)
            x0 = krDeb*(b + delta)

      IF (sh.eq.3) then
        theta = 3.d0/4.d0*Pi
        m = (c+2*b*cos(theta))/(c+b*cos(theta))
        g_ion_nlin = Evans1984(m, x0, s) /s
!       g_ion_nlin = DUM_Ohshima1982(m, x0, s)/s
!                   g_ion_nlin = Ohshima_g_diff(m, x0, s)
!           print*,'NONLINEAR',g_ion_nlin,m,s,achrg,krDeb,x0

      ELSE
      IF (sh.eq.2) then
        g_ion_nlin = DUM_Ohshima1982(2.d0, x0, s)/s    ! or  Ohshima_g_diff(2.0d0, x0, s)
!                   g_ion_nlin = Ohshima_g_diff(2.0d0, x0, s)
            ELSE
      IF (sh.eq.1) then
        g_ion_nlin = Ohshima_g_diff(1.0d0, x0, s)                    !    or      Evans1984(1.d0,    x0,    s)/s
Ohshima_g_diff(1.0d0, x0, s)
!     g_ion_nlin = DUM_Ohshima1982(1.d0, x0, s)/s
            ELSE
      IF (sh.eq.0) then
!                   print*, 'sh=',sh,alfa,b,c,delta,'Deb=',krDeb,s

        g_ion_nlin = Evans1984(0.d0, x0, s)/s
!       g_ion_nlin = DUM_Ohshima1982(0.d0, x0, s)/s
!           g_ion_nlin = Ohshima_g_diff(0.0d0, x0, s)
      ELSE
            write(*,*) "Error in g_ion_nlin()"
      ENDIF
```

```
      ENDIF
      ENDIF
      ENDIF
      RETURN
      END
!C*****************************************************************
      REAL*8 function g_zwitter(alfa,b,c,sh)
      IMPLICIT none
          REAL*8 b, c, sh, alfa
      REAL*8 m, theta, alfag_dip, delta_dip,a_dip,adelta,l_B,&
            m1,coef,Ddip,Rdip
      integer, parameter :: ch = 9
      real*8 MconstA(ch), MconstB(ch)
             ! 1     2     3     4     5     6     7     8      9
             ! Nc    Vs    ls    a0    ap   delta deltaH  logic  ddip
      REAL*8 T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
      epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,&
            CsurA,CsurB
      COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,&
     epsilon,MconstA, MconstB,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd, &
     CsurA,CsurB

          call alf_dip(alfa,alfag_dip,delta_dip,Ddip)          ! given fraction of A surfactant  calculate
fraction of dipolar molecules and location of dipole and transfer the dipole length Ddip
    !   write(*,*)'alfa= ',alfa,'alfa_dip= ',alfag_dip,'sh=',sh
          if (alfag_dip<1.0d-15.or.Ddip==0.) then                    ! no dipole moments
in the surfactant system
          g_zwitter = 0.0d0
          return
          endif
          a_dip=adelta(sh,alfa,b,c,delta_dip)                  ! calculate surface per molecule at
location of dipoles

          l_B = e**2/(epsilon*k*T)*1.0d8                       ! [A]   CGSE units
            coef=alfag_dip*2.0d0*l_B/a_dip                     ! here a_dip/alfag_dip gives
the surface per dipolar molecule at location of dipoles
            Rdip = b + delta_dip

      IF (sh.eq.3) then
       theta = 3.d0/4.d0*pi
       m = (c+2*b*cos(theta))/(c+b*cos(theta))
          m1=m-1.0d0
!         write(*,*) "m=", m
       g_zwitter=coef*Rdip*( 1.0d0-(Rdip/(Rdip+Ddip))**m1 )/m1  !toroidal capacitor
      ELSE
```

```
      IF (sh.eq.2) then
        g_zwitter = coef*Rdip*Ddip/(Rdip + Ddip)      ! spherical capacitor
            ELSE
      IF (sh.eq.1) then
       g_zwitter = coef*Rdip*dlog(1.0d0+Rdip/Ddip)    ! cylindric capacitor
      ELSE
      IF (sh.eq.0) then
       g_zwitter = coef*Ddip                                    ! planar capacitor
      ELSE
            write(*,*) "Error in g_ion_nlin()"
      ENDIF
      ENDIF
      ENDIF
      ENDIF
      RETURN
      END
!C*****************************************************************
      subroutine DUM_Ohshima_f(sh, y0, x0, f, df)
      implicit none
      real*8 sh, y0, x0, f, df
!c    On entry:
!c    y0 - variable
!c    x0, sh - parameters
!c    On exit:
!c    f, df - function and its derivative
      real*8 t
      t  = dtanh(0.25*y0)
      f  = 2.0*dsinh(0.5*y0)+2.0*sh/x0*t
      df =     dcosh(0.5*y0)+0.5*sh/x0*(1.0-t*t)
      end
!C*****************************************************************
      real*8 function DUM_Ohshima1982(sh, x0, s)
      implicit none
      real*8 sh, x0, s
!c    On entry:
!c    sh - shape factor, 2 for spheres and 1 for cylinders
!c    x0 - reduced radius of the surface (x0=kappa*R)
!c    s  - reduced surface charge density
!c
!c    On exit:
!c    returns integral \int_0^s y_0ds
!c
      real*8 f, df, dy0, y0, z, integral
      z=1.0d0+sh/x0
```

```
      z=dsqrt(z*z+0.25*s*s)-sh/x0
!c    Initial estimation according to Evans et al. 1984
      y0=2.*dlog(z+dsqrt(z*z-1.))
      do
       call DUM_Ohshima_f(sh,y0,x0,f,df)
       dy0 =-(f-s)/df
       y0  = y0+dy0
!c       write (*,*) y0, "  ", f
       if(dabs(dy0) < 1.0d-7) exit
      enddo
      integral = 4.0*dcosh(0.5*y0)+8.0*sh/x0*dlog(dcosh(0.25*y0))-4.0
      DUM_Ohshima1982 = s*y0 - integral
      end function
!*****************************************************************
subroutine Ohshima_f(sh, y0, x0, f, df)
implicit none
real*8 sh, y0, x0, f, df
! On entry:
! y0 - variable
! x0, sh - parameters
! On exit:
! f, df - function and its derivative
real*8 t
t  = dtanh(0.25*y0)
f  = 2.0*dsinh(0.5*y0)+2.0*sh/x0*t
df =     dcosh(0.5*y0)+0.5*sh/x0*(1.0-t*t)
end
!*********************************************************************
real*8 function Ohshima_g_diff(sh, x0, s)
implicit none
real*8 sh, x0, s,nom
!
! On entry:
! sh - shape factor, 2 for spheres and 1 for cylinders
! x0 - reduced radius of the surface (x0=kappa*R)
! s  - reduced surface charge density
!
! On exit:
! returns reduced electric free energy per CHARGE
!
real*8 f, df, dy0, y0, z, integral
!print *, 'Ohshima'
z=1.+sh/x0
nom = 1.
```

```
z=dsqrt(z*z+0.25*s*s)-sh/x0
! print *, sh,x0, s, z
! Начальное приближение по Эвансу
y0=2.*dlog(z+dsqrt(z*z-1.))
do
  call Ohshima_f(sh,y0,x0,f,df)
  dy0 =-(f-s)/df
  y0  = y0+dy0
 !   write (*,*) y0, "  ", f
  if(dabs(dy0) < 1.0d-6) exit
  if (nom>1.d+3) exit
  nom = nom + 1.
enddo
integral = 4.0*dcosh(0.5*y0)+8.0*sh/x0*dlog(dcosh(0.25*y0))-4.0
Ohshima_g_diff = y0 - integral/s
end function
! ************************************************************
      real*8 function Evans1984(m, x0, s)
      implicit none
      real*8 m, x0, s
!c     On entry:
!c     m - shape factor
!c     x0 - reduced radius of the surface (x0=kappa*R)
!c     s  - reduced surface charge density
!c     On exit:
!c     returns integral \int_0^s y_0ds
            REAL*8 z, y0
            if( s < 0.0) then
              write(*,*) "error in Evans1984(): s < 0"
              stop
            endif
       z=dsqrt((1.d0+m/x0)**2+0.25d0*s**2)-m/x0
!           print*,'m= ',m,x0,s,'z= ',z,(dsqrt((1.d0+m/x0)**2)-m/x0)

             If(z.LE.1.d0)then
             Evans1984=0.d0
             return
             endif

      y0=2.d0*dlog(z+dsqrt(z*z-1.d0))
            Evans1984=          &
            y0*s-2.0*s*dsqrt((z+1.0)/(z-1.0)) + &
            8.0*dsqrt(1.0+m/x0)-8.0*m/x0*                &
            (dlog(dsqrt(z+1.0)+s/(2.0*dsqrt(z-1.0))) &
```

```
   -dlog(dsqrt(2.d0)+dsqrt(2.0+2.0*m/x0)))

!c       g_el10 := 2*darccosh(z)-2*dsqrt((z+1)/(z-1))+8/s*dsqrt(1+m/x0)-
!c      8*m/(x0*s)*(dlog(dsqrt(z+1)+s/(2*sqrt(z-1)))-dlog(dsqrt(2.)+dsqrt(2+2*m/x0)));

!C 2.0*(dlog(s/2.0+dsqrt(1.0+s*s/4.0)) +
!C> (1-sqrt(1+s*s/4.0))*2.0/s -
!C> 2*m/(x0*s)*dlog((1.0+dsqrt(1+s*s/4.0))/2.0))
          return
          END
!C*********************************************************************
     real*8 function Evans1983(m, x0, s)                                  ! exact for plane
     implicit none
     real*8 m, x0, s

          Evans1983 = s*2.0d0*(dlog(s/2.0+dsqrt(1.0d0+s*s/4.0d0))+ &
               (1-dsqrt(1+s*s/4.0d0))*2.0d0 -  &
          2.0d0*m/(x0)*dlog((1.0d0+dsqrt(1+s*s/4.0d0))/2.0d0))
     return
          end function
!C*********************************************************************
real*8 function fcyl (gg)
implicit none
real*8 gg(1,2), X1A,X1B,alfaA1,chpotcyl
real*8 alfa,R,RUbound,lsinf,lssup
common /monomers/ X1A,X1B,alfaA1
common /lsformin/ lsinf,lssup

if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15)then
gg(1,2)=alfaA1
fcyl = chpotcyl(gg)
return
endif

alfa = gg(1,2)
R = gg(1,1)
!if((R>=RUbound(alfa)).or.(R<lsinf)) then
                                                                    ! add condition on alfa
.or.(alfa>=1.0d0)
!fcyl = 1.0d3*((R-RUbound(alfa))/(R+RUbound(alfa)))**2 +1.0d3 ! + 5.0d1*((alfa-0.5d0)/(alfa+0.5d0))**2
          ! original fcyl = 1.0d3*((R-3.0d0*lsinf)/(R+3.0d0*lsinf))**2 +1.0d3 + 5.0d1*((alfa-
0.5d0)/(alfa+0.5d0))**2
!return
!endif
!fcyl = -(X1A**gg(1,2))*(X1B**(1.0d0 - gg(1,2)))*dexp(- chpotcyl(gg))
```

```
fcyl = -alfa*dlog(X1A)-    (1.0d0-alfa)*dlog(X1B)+chpotcyl(gg)
!print*,'cyl  ', fcyl
return
end function
! ************************************************************
real*8 function fsph (gg)
implicit none
real*8 gg(1,2)
real*8 alfacylopt,Rcylopt,gcylopt, R, alfa,chpot,nsph,J,lnK,vs
common /optcyl/ alfacylopt,Rcylopt,gcylopt
real*8 X1A,X1B,alfaA1
real*8 RUbound,lsinf,lssup
common /monomers/ X1A,X1B,alfaA1
common /lsformin/ lsinf,lssup

if (alfaA1<1.0d-15.or.dabs(alfaA1-1)<1.0d-15)then
gg(1,2)=alfaA1
alfa = gg(1,2)
R = gg(1,1)
nsph = 4./3.*3.1415926*R**3/vs(alfa)
lnK = nsph*(chpot(gg) - gcylopt)
fsph = lnK
return
endif

alfa = gg(1,2)
R = gg(1,1)
nsph = 4./3.*3.1415926*R**3/vs(alfa)
J = nsph*(alfa - alfacylopt)
lnK = nsph*(chpot(gg) - gcylopt)
!write (*,*) 'X1A/B sph=',X1A,X1B
!fsph = -(X1A**J)*(X1B**(1.d0 - J))*dexp(- lnK)              ! inf cycling with NLPB: might be necessary to turn on the next string

!if((R>=RUbound(alfa))) then
                                                            ! add condition on alfa .or.(alfa>=1.0d0)
!fsph = 1.0d3*((R-RUbound(alfa))/(R+RUbound(alfa)))**2 +1.0d3 ! + 5.0d1*((alfa-0.5d0)/(alfa+0.5d0))**2
!return
!elseif(R<lsinf)    then
!fsph = 1.0d3*((R-0.2d0*RUbound(alfa))/(R+0.2d0*RUbound(alfa)))**2 +1.0d3
!return
!endif
fsph = - (J*dlog(X1A) + (1.0d0- J)*dlog(X1B) - lnK)          ! overflow c LPB, works  with NLPB
 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
!fsph = -(X1A**gg(1,2))*(X1B**(1. - gg(1,2)))*dexp(- chpot(gg))       ! similar but easier
return
end function
!**********************************************************************
real*8 function fsph_max (gg)
implicit none
real*8 gg(1,2)
real*8 R, alfa,chpot,nsph,vs
real*8 X1A,X1B,alfaA1
common /monomers/ X1A,X1B,alfaA1

R = gg(1,1)

if (alfaA1<1.0d-15.or.dabs(alfaA1-1.0d0)<1.0d-15)then                                    !
pure A  or B
gg(1,2)=alfaA1
alfa = gg(1,2)
nsph = 4./3.*3.1415926*R**3/vs(alfa)
fsph_max = nsph*(chpot(gg) - dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1)))          !          alfaA1          toggles
between X1A  and X1B for pure components
return
endif
!               MIXTURES A+B:
alfa = gg(1,2)
nsph = 4./3.*3.1415926*R**3/vs(alfa)
fsph_max = nsph*(chpot(gg) - alfa*dlog(X1A)-(1.0d0-alfa)*dlog(X1B))
return
end function

!**************************************************************************
real*8 function fsph_gmax (gg)
implicit none
real*8 gg(1,2)
real*8 R, alfa,chpot,nsph,vs
real*8 X1A,X1B,alfaA1
common /monomers/ X1A,X1B,alfaA1

R = gg(1,1)

if (alfaA1<1.0d-15.or.dabs(alfaA1-1.0d0)<1.0d-15)then                                    !
pure A  or B
gg(1,2)=alfaA1
alfa = gg(1,2)
nsph = 4./3.*3.1415926*R**3/vs(alfa)
```

```
      fsph_gmax = nsph*(chpot(gg) - dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1)))-dlog(nsph)   !          alfaA1
toggles between X1A  and X1B for pure components
      return
      endif
      !              MIXTURES A+B:
      alfa = gg(1,2)
      nsph = 4./3.*3.1415926*R**3/vs(alfa)
      fsph_gmax = nsph*(chpot(gg) - alfa*dlog(X1A)-(1.0d0-alfa)*dlog(X1B))-dlog(nsph)
      return
      end function
      !**********************************************************************
      !          type of surfactants: 1 - cationic, 2 - anionic, 3 - nonionic  (if Ddip>0 zwitterionic)
        SUBROUTINE fcn_pl(n,x,f)
        IMPLICIT none
        INTEGER n
            REAL*8 lsinf,lssup,X1A,X1B,alfaA1
            COMMON/alfabound/right_alfa,left_alfa
            common /lsformin/ lsinf,lssup
            common /monomers/ X1A,X1B,alfaA1
            REAL*8 right_alfa,left_alfa
        REAL*8 x(n), f, g_pl,x1,x2,bright,bleft,RUbound
        EXTERNAL g_pl

            x1=x(1)*lssup
            x2=x(2)
        bright=RUbound(x2)
        bleft =0.3d0*bright
            if (x1.GT.bright) then
            x1=bright
            f=1000.0d0
            return
            end if
            if (x1.LT.bleft) then
            x1=bleft
            f=1000.0d0
            return
            end if
            if (x2.GT.right_alfa) then
            x2=right_alfa
            f=1000.0d0
            return
            end if
            if (x2.LT.left_alfa) then
            f=1000.0d0
```

```
            x2=left_alfa
            return
            end if
            f=g_pl(x1,x2)
      RETURN
      END
!C*********************************************************************
      SUBROUTINE fcn_pl_pure(m,x,f)
      IMPLICIT none
      INTEGER m
            REAL*8 lsinf,lssup,X1A,X1B,alfaA1
!           common /lsformin/ lsinf,lssup
            common /monomers/ X1A,X1B,alfaA1
      REAL*8 x(m), f, g_pl,x1,x2,bright,bleft,RUbound
      EXTERNAL g_pl

      bright=RUbound(alfaA1)
      bleft =0.3d0*bright

            x1=x(1)*bright
            x2=alfaA1
            if (x1.GT.bright) then
            x1=bright
            f=1000.0d0
            return
            end if
            if (x1.LT.bleft) then
            x1=bleft
            f=1000.0d0
            return
            end if
            f=g_pl(x1,x2)
      RETURN
      END
!C*********************************************************************
            SUBROUTINE fcn_tor(n,x,f)
      IMPLICIT none
      INTEGER n
      REAL*8 x(n), f, g_tor,x1,x2,x3,X1A,X1B,alfaA1
            REAL*8 lssup,lsinf,right_alfa,left_alfa,bleft,bright,RUbound
            COMMON/alfabound/right_alfa,left_alfa
            common /lsformin/ lsinf,lssup
            common /monomers/ X1A,X1B,alfaA1
      EXTERNAL g_tor
```

```
      x1=x(1)*lssup
      x2=x(2)*lssup
      x3=x(3)
       bright=RUbound(x3)
   bleft =0.3d0*bright

      if (x1.GT.bright) then
      x1=bright
      f=1000.0d0
      return
      end if
      if (x1.LT.bleft) then
      x1=bleft
      f=1000.0d0
      return
      end if
      if (x3.GT.right_alfa) then
      x3=right_alfa
      f=1000.0d0
      return
      end if
      if (x3.LT.left_alfa) then
      f=1000.0d0
      x3=left_alfa
      return
      end if
      if ((x2/x1).LT.1.10d0) then
      x2=1.10d0*x1
      f=1000.0d0
      return
      endif
      if ((x2/x1).GT.100.0d0) then
      x2=100.0d0*x1
      f=1000.0d0
      return
      endif
      f=g_tor(x1,x2,x3)
   RETURN
   END
!C*****************************************************************
      SUBROUTINE fcn_jun(n,x,f)
   IMPLICIT none
   INTEGER n,depend
```

```
REAL*8 x(n), f, g_jun,g_jun2,x1,x2,x3,X1A,X1B,alfaA1
      REAL*8 lssup,lsinf,right_alfa,left_alfa,bleft,bright,RUbound
      COMMON/alfabound/right_alfa,left_alfa
      common /lsformin/ lsinf,lssup
      common /monomers/ X1A,X1B,alfaA1
COMMON/control/depend
EXTERNAL g_jun, g_jun2

      x1=x(1)*lssup
      x2=x(2)*lssup              !x2 - c
      x3=x(3)                          !x3 - alfa in torus  or  other, see g_jun

bright=RUbound(x3)
bleft =0.3d0*bright

      if (x1.LT.bleft) then
      x1=bleft
      f=1000.0d0
      return
      endif
      if (x1.GT.bright) then
      x1=bright
      f=1000.0d0
      return
      endif
      if (x3.LT.left_alfa) then
      x3=left_alfa
      f=1000.0d0
      return
      end if
      if (x3.GT.right_alfa) then
      x3=right_alfa
      f=1000.0d0
      return
      endif
      if ((x2/x1).LT.1.30d0) then                    !!!!!!!!!!!!!!!!!!!!!!  originally
was 1.1
      x2=1.30d0*x1
      f=1000.0d0
      return
      endif
      if ((x2/x1).GT.100.0d0) then
      x2=100.0d0*x1
```

```
                f=1000.0d0
                return
                endif
                if(depend) then
            f = g_jun(x1,x2,x3)                    !works when depend=1

          !f  = -x3*dlog(X1A)-     (1.0d0-x3)*dlog(X1B)+g_jun(x1,x2,x3)          ! here  x3 is not exactly  the
average composition - trial
                else
            f = g_jun2(x1,x2,x3)                !works when depend=0

          !f  = -x3*dlog(X1A)-     (1.0d0-x3)*dlog(X1B)+g_jun2(x1,x2,x3)          ! here  x3 is not exactly  the
average composition - trial

           !        write(*, '(A9,E14.5,2(2X,F12.7),E14.5)') 'fcnjun = ',f, x1,x2,x3
            endif
            RETURN
            END
          !C*****************************************************************
                SUBROUTINE fcn_junMAX(n,x,f)
            IMPLICIT none
            INTEGER n,depend

            REAL*8 x(n), f, g_jun,g_jun2,x1,x2,x3,X1A,X1B,alfaA1
                REAL*8
lssup,lsinf,right_alfa,left_alfa,bleft,bright,RUbound,r_patch,alfa_pl,alf_JUN,Jun_alf,g_patch,n_tor,n_pl,njun
                COMMON/alfabound/right_alfa,left_alfa
                common /lsformin/ lsinf,lssup
                common /monomers/ X1A,X1B,alfaA1
                COMMON/optpl/r_patch,alfa_pl,g_patch
            COMMON/control/depend
            EXTERNAL g_jun, g_jun2

                x1=x(1)*lssup
                x2=x(2)*lssup              !x2 - c
                x3=x(3)                            !x3 - alfa in torus  or  other, see g_jun

            bright=RUbound(x3)
            bleft =0.3d0*bright

                if (x1.LT.bleft) then
                x1=bleft
                f=1000.0d0
                return
```

```
                endif
                if (x1.GT.bright) then
                x1=bright
                f=1000.0d0
                return
                endif
                if ((x2/x1).LT.1.30d0) then                    !!!!!!!!!!!!!!!!!!!!!!  originally
was 1.1
                x2=1.30d0*x1
                f=1000.0d0
                return
                endif
                if ((x2/x1).GT.100.0d0) then
                x2=100.0d0*x1
                f=1000.0d0
                return
                endif
        !               pure A  or B
        !       if (alfaA1<1.0d-15.or.dabs(alfaA1-1.0d0)<1.0d-15)then

        !x3=alfaA1
        !if(depend) then
        !f  =  -dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1))+g_jun(x1,x2,x3)       !works when depend=1   alfaA1
toggles between X1A  and X1B for pure components
        !       else
        !f  =  -dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1))+g_jun2(x1,x2,x3)                !works          when
depend=0          alfaA1  toggles between X1A  and X1B for pure components
        !   endif
        !return
        !endif
        !                MIXTURES A+B:
                if (x3.LT.left_alfa) then
                x3=left_alfa
                f=1000.0d0
                return
                end if
                if (x3.GT.right_alfa) then
                x3=right_alfa
                f=1000.0d0
                return
                endif
        call Av_alf_jun(x1,x2,r_patch,alfa_pl,x3,alf_JUN,n_tor,n_pl,njun)
        !alf_JUN = Jun_alf(x1,x2,r_patch,alfa_pl,x3)                ! average fraction in junction

                if(depend) then
```

```
      f  = (-alf_JUN*dlog(X1A)-        (1.0d0-alf_JUN)*dlog(X1B)+g_jun(x1,x2,x3))*njun              -dlog(njun)
!works when depend=1
            else
      f  = (-alf_JUN*dlog(X1A)-        (1.0d0-alf_JUN)*dlog(X1B)+g_jun2(x1,x2,x3))*njun -dlog(njun)
              !works when depend=0
          endif
          RETURN
          END
      !C*********************************************************************
            SUBROUTINE fcn_jun_pure(m,x,f)
          IMPLICIT none
          INTEGER m,depend

          REAL*8 x(m), f, g_jun,g_jun2,x1,x2,x3,X1A,X1B,alfaA1
            REAL*8 lssup,lsinf,bleft,bright, RUbound
      !     common /lsformin/ lsinf,lssup
          COMMON/control/depend
            common /monomers/ X1A,X1B,alfaA1
          EXTERNAL g_jun, g_jun2

          bright=RUbound(alfaA1)
          bleft =0.3d0*bright
            x1=x(1)*bright
            x2=x(2)*bright
            x3=alfaA1

            if (x1.LT.bleft) then
            x1=bleft
            f=1000.0d0
            return
            endif
            if (x1.GT.bright) then
            x1=bright
            f=1000.0d0
            return
            endif

            if ((x2/x1).LT.1.30d0) then                                    !!!!!!!!!!!!!!!!!!!!!!  originally
was 1.1
            x2=1.30d0*x1
            f=1000.0d0
            return
            endif
            if ((x2/x1).GT.200.0d0) then                    ! was 100
```

```
            x2=100.0d0*x1
            f=1000.0d0
            return
            endif
            if(depend) then
       f = g_jun(x1,x2,x3)                    !works when depend=1
            else
       f = g_jun2(x1,x2,x3)          !works when depend=0
 !          write(*, '(A9,E14.5,2(2X,F12.7),E14.5)') 'fcnjun = ',f, x1,x2,x3
       endif
       RETURN
       END
      !C*******************************************************************************
            SUBROUTINE fcn_jun_pureMAX(m,x,f)
       IMPLICIT none
       INTEGER m,depend
       REAL*8 x(m), f, g_jun,g_jun2,x1,x2,x3,X1A,X1B,alfaA1, r_patch,alfa_pl, g_patch,alf_JUN, n_tor,n_pl,
njun
            REAL*8 lssup,lsinf,bleft,bright, RUbound
     COMMON/control/depend
            common /monomers/ X1A,X1B,alfaA1
            COMMON/optpl/r_patch,alfa_pl,g_patch
       EXTERNAL g_jun, g_jun2
       bright=RUbound(alfaA1)
       bleft =0.3d0*bright
            x1=x(1)*bright
            x2=x(2)*bright
            x3=alfaA1

            if (x1.LT.bleft) then
            x1=bleft
            f=1000.0d0
            return
            endif
            if (x1.GT.bright) then
            x1=bright
            f=1000.0d0
            return
            endif
            if ((x2/x1).LT.1.30d0) then
            x2=1.30d0*x1
            f=1000.0d0
            return
            endif
```

```
                if ((x2/x1).GT.200.0d0) then                                    ! was 100
                x2=100.0d0*x1
                f=1000.0d0
                return
                endif
        call Av_alf_jun(x1,x2,r_patch,alfa_pl,x3,alf_JUN,n_tor,n_pl,njun)
        if(depend) then
        f   =  (-dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1))+g_jun(x1,x2,x3))*njun  -dlog(njun)          !works when
depend=1          alfaA1  toggles between X1A  and X1B for pure components
                else
        f  =  (-dlog(X1A*alfaA1+X1B*(1.0d0-alfaA1))+g_jun2(x1,x2,x3))*njun       -dlog(njun)
!works when depend=0       alfaA1  toggles between X1A  and X1B for pure components
            endif
        return
        end
        !**********************************************************************
        SUBROUTINE f2ves(n,x,f)
         IMPLICIT none
         INTEGER n
        Common /ves_dumpol/r_ves
        Common /ves_dumpol1/alfao,alfai
        REAL*8 lssup,lsinf,right_alfa,left_alfa,ls_min,alfao
        COMMON/alfabound/right_alfa,left_alfa
        common /lsformin/ lsinf,lssup
        integer, parameter :: ch = 9
        real*8 MconstA(ch), MconstB(ch)
        REAL*8   T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB
        real*8 epsilon,nzero,krDeb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
        COMMON /A/ T,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB, &
        epsilon,MconstA, MconstB, &
        nzero,krDEb,dchpotCH2,dchpotCH3,Cadd,CsurA,CsurB
        common /monomers/X1A,X1B,alfaA1
        real*8 X1A,X1B,alfaA1
        REAL*8 r_ves,alfa,alfai,agg_vo,agg_veso,agg_v,agg_ves
        REAL*8 x(n), f, g_ves, x1,x2,x3,x4,x5,x6,C,agg_vi,agg_vesi
        EXTERNAL g_ves
        Common /tauotaui/C
          ls_min=lsinf/0.3d0
                left_alfa = 1.0d-14             !0.0d0
          right_alfa = 1.0d0-1.0d-14        !1.0d0
                x1=r_ves
                x2=x(1)*ls_min/(1+C)                !tauo
                x3=C*x2
                x4=alfao                                                !alfao
```

```
x5=alfai                                !alfai

      agg_vo=agg_veso(r_ves,x2,x3,x4)
      agg_v=agg_ves(r_ves,x2,x3,x4,x5)
      alfa = (x4*agg_vo+ x5*agg_vi)/agg_v

      if (x2.GT.ls_min) then
      x2=ls_min
      f=1000.0d0
      return
      end if
      if (x2.LT.lsinf) then
      x2=lsinf
      f=1000.0d0
      return
      end if
      if (x3.GT.ls_min) then
      x3=ls_min
      f=1000.0d0
      return
      end if
      if (x3.LT.lsinf) then
      x3=lsinf
      f=1000.0d0
      return
      end if
      if (x5.GT.right_alfa) then
      x5=right_alfa
      f=1000.0d0
      return
      end if
      if (x5.LT.left_alfa) then
      f=1000.0d0
      x5=left_alfa
      return
      end if
      if (x4.GT.right_alfa) then
      x4=right_alfa
      f=1000.0d0
      return
      end if
      if (x4.LT.left_alfa) then
      f=1000.0d0
      x4=left_alfa
```

```
            return
            end if
            f=-alfa*dlog(X1A)-(1.0d0-alfa)*dlog(X1B)+g_ves(x1,x2,x3,x4,x5)
          RETURN
          END
        ! ***************************Numerical ****************
        subroutine coordsquare(FF,Xzero,a,b,epsab,c,d,epscd,XXeps,minXX,minZ)
        ! минимизация f = FF(XX), где XX =( XX(1,1), XX(1,2) ) - вектор
        ! XX(1,1) принадлежит [a,b]
        ! XX(1,2) принадлежит [c,d]
        ! epsab - точность поиска по первой составляющей вектора XX
        ! epscd -   -- // --         второй     -- // --
        ! XXeps - точность определения минимума
        ! ( одномерный поиск - метом золотого сечения)
        ! minXX,minZ - выходные параметры
        implicit none
        real*8 logicalcoord,Xzero(1,2),minXX(1,2),minZ,XX(1,2),X0(1,2),delX(1,2)
        real*8 a,b,epsab,c,d,epscd,XXeps(1,2)
        real*8 FF,miny,minx,bright,RUbound,alfa,aleft
        external FF,RUbound
        common /coordsquarecom / logicalcoord,XX
        X0  = Xzero
        do
          XX = X0
          logicalcoord = 1.
          call golden (FF,c,d,epscd,minx,miny)
          XX(1,2)  = minx
          alfa = minx
          logicalcoord = 2.
           ! R-optimization
        if (b>RUbound(alfa)) then
        bright=RUbound(alfa)
        else
        bright=b
        endif
        if (a<bright*0.4d0) then                                                    !
controlling minimal  R  -  excluding premicellar  aggregates
        aleft= bright*0.4d0
        else
        aleft=a
        endif
          call golden (FF,aleft,bright,epsab,minx,miny)
          XX(1,1)  = minx
          delX = XX - X0
```

```
 if ((dabs(delX(1,1))<= XXeps(1,1)).and.(dabs(delX(1,2))<= XXeps(1,2)))   exit
 X0 = XX
end do
minXX = XX
minZ = FF(XX)
end subroutine
!**************************************************************************
real*8 function f(x,FF)
implicit none
real*8 x,XX(1,2),logicalcoord,ff
external ff
common /coordsquarecom / logicalcoord,XX
if (logicalcoord == 1.) then
XX(1,2) = x
f = FF(XX)
else
XX(1,1) = x
f = FF(XX)
end if
end function
!**************************************************************************
subroutine golden (FF,left,right,eps,minx,miny)
implicit none
real*8 ff,a,b,eps,minx,miny,x1,x2,y1,y2,h,left,right,f
external ff,f
! нахождение минимума функции одной переменной
! методом золотого сечения
! left,right,eps - входные параметры
! minx,miny - выходные параметры
a = left
b = right
x1 = a + 0.382*(b - a)
x2 = a + 0.618*(b - a)
y1 = f(x1,ff)
y2 = f(x2,ff)
do
if(y1>y2) then                                                    ! try to include equality
 a = x1
 h = (b - a)/2.
 if(h<= eps) exit
 x1 = x2
 y1 = y2
 x2 = a + 0.618*(b - a)
 y2 = f(x2,ff)
```

```
elseif (y1<y2) then
  b = x2
  h = (b - a)/2.
  if(h<= eps) exit
  x2 = x1
  y2 = y1
  x1 = a + 0.382*(b - a)
  y1 = f(x1,ff)
else
  a = x1
  b = x2
  h = (b - a)/2.
  if(h<= eps) exit
  x1 = a + 0.382*(b - a)
  x2 = a + 0.618*(b - a)
  y1 = f(x1,ff)
  y2 = f(x2,ff)
end if
!  print *, 'a==',a,'b==',b
end do
minx = (a + b)/2.
miny = f(minx,ff)
end subroutine
!**********************************************************************
subroutine dihotom (a, del, eps,minx,miny)
implicit none
real*8 a, del, eps,minx,miny
real*8 x1,x2,y1,y2,chag
logical Ybig1
! debugging:
real*8 X1A,X1B,alfaA1
common /monomers/ X1A,X1B,alfaA1
 !CMC (X1,Ybig1,CMCfun)
! одномерная минимизация методом
! дихотомии
! a - начальная точка
! del - веществ. число,
! начальная величина шага
! a, del, eps - входные параметры
! minx,miny - выходные параметры
chag = del
x1 = a
x2 = x1 + chag
do
```

```
  if(dabs(x1 - x2)<eps) exit
  call CMC (x1,Ybig1,y1)
! print *,'1 olya!! ', x1,y1,chag,Ybig1,alfaA1
  if (Ybig1 .eqv..true.) then
     if(x1<=dabs(2*chag)) chag = chag/2.d0
     x1 = x1 + chag
     x2 = x1 + chag
           cycle
   end if
  call CMC (x2,Ybig1,y2)
!print *,'2 oppa!! ', x2,y2,chag,Ybig1
    if(y1>y2) then
    if(x2<=2*dabs(chag)) chag = chag/2.d0
    x1 = x2
    x2 = x1 + chag
  else
    x1 = x2
    chag = - chag/2.d0
    x2 = x1 + chag
  end if
end do
miny = y2
minx = (x1 + x2)/2.d0
return
end subroutine
!**********************************************************************
subroutine indsurf(H,VheadA,VheadB,VtailA,VtailB,CH2lgth,ntailA,ntailB)
! divides surfactant A, surfactant B into head and tail portion
! assignes H-shell thickness and functional groups to surfA, surfB and solvent(water)
IMPLICIT REAL*8 (A-H,O-Z)
integer, parameter :: ch = 9
real*8 MconstA(ch), MconstB(ch)
!   1     2     3     4     5     6     7      8       9
!  Nc     Vs    ls    a0    ap   delta deltaH  logic    ddip
REAL*8 T2,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,&
    nzero,krDeb,dchpotCH2,dchpotCH3,Cadd
COMMON /A/ T2,Pi,psi,k,e,Nav,sigmaw,sigmaA,sigmaB,sigmasA,sigmasB,epsilon,MconstA, MconstB,&
     nzero,krDeb,dchpotCH2,dchpotCH3,Cadd
COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
COMMON /B/ W(10,10),WT(10,10),WTT(10,10),REDT(10,10)
COMMON /C/ QT,RT,T,TT,R(10),RS(10,10),Q(10),QS(10,10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
           TT=298.16d0
```

```
        CH3lgth = 1.50
        CH2lgth = 1.265
        OHlgth = 1.17
        CMIMlgth = 11.0      !10.02 (Cl)   !6.02

        volCH3 = 54.6 + 0.124*(T - 298.)
        volCH2 = 26.9 + 0.0146*(T - 298.)
        volOH=25.0*1.0076
        volMIM=25.0*5.1893

        nAch2=1.0
        nBch2=3.0
        PolAlgth= nAch2*CH2lgth+CMIMlgth
        PolBlgth= nBch2*CH2lgth+OHlgth

        volPolarA=volMIM+nAch2*volCH2
        volPolarB=volOH+nBch2*volCH2
! solvent-water:
        vsolv=30.0

        nheadA=1.0   !(only ch2-groups count)
        nheadB=2.0

        NcA=MconstA(1)
        NcB=MconstB(1)
        ntailA=NcA-nheadA   ! all polar groups in the head: 1-polar group and several repaeted CH2 groups
        ntailB=NcB-nheadB
        nheadA=nheadA+1
        nheadB=nheadB+1
        VheadA=add(nheadA,volCH2,volMIM)
        VheadB=add(nheadB,volCH2,volOH)
        H=add(nheadA,CH2lgth,CMIMlgth)
        VtailA=add(ntailA,volCH2,volCH3)
        VtailB=add(ntailB,volCH2,volCH3)
        NK=3.0
        NG= 7.0
        ZZ=10.0
        ZR=0.5*ZZ

        do I=1,NG
        do J=1,NG
        W(I,J)=0.0d0
        WT(I,J)=0.0d0
        WTT(I,J)=0.0d0
```

```
            end do
            end do

! GROUP #   1     2     3     4       5         6      7
! GROUPS   CH3    CH2   O(OH)   H(OH)  O(water)  2H(water)  MIM

            do 1300 I=1,NK
            do 1300 J=1,NG
            W(J,J)=0.0
! if there's no group in the molecule then Rs=0 and Ls=1
            RS(I,J)=0.0
 1300       BLIS(I,J)=1.0
! surfactant A (MIM)  - head partion
            RS(1,1)= 0.0d0                  !1.9621
            BLIS(1,1)=1.0d0                 !0.5
            RS(1,2)=nAch2*1.0865
            RS(1,7)= 3.1893               !3.1893      works
            BLIS(1,7)=0.5

! propanol
            RS(2,1)=0.0d0             !1.9621
            BLIS(2,1)=1.0d0           !0.5
            RS(2,2)=nBch2*1.0865
            RS(2,3)=1.0076
            BLIS(2,3)=0.9
            RS(2,4)=0
            BLIS(2,4)=0.5

! solvent : water
            RS(3,5)=1.1903
            BLIS(3,5)=1.0
            RS(3,6)=0.0
            BLIS(3,6)=0.0

! CH3- other groups:
            W(1,2)=0.0256
            W(1,3)=0.2639
            W(1,4)=0.2639
            W(1,5)=0.0345
            W(1,6)=0.0345
            W(1,7)=-0.45
! CH2- other groups:
            W(2,3)=0.0345     !as w.water         !0.2639                    original
            W(2,4)=0.0345     !as w.water         !0.2639                    original
```

```
                W(2,5)=0.0345
                W(2,6)=0.0345
                W(2,7)=-0.45                    !   original  -0.45                    ! weak repulsion 0.03
        ! O(OH)- other groups:
                W(3,4)=-4.2668
                W(3,6)=-4.2693
                W(3,7)=-5.5      ! original   -3.50                                    ! strong attraction  -5.50 !          weak
attraction   -       0.5                    ! try ysical: repulsion 2.0

        ! H(OH)- other groups:
                W(4,5)=-5.7439
                W(4,7)=0.0                              ! 0.0   original       2.0 repulsion     -4.5   attraction
        ! O(H2O)- other groups:
                W(5,6)=-4.988
                W(5,7)=-3.5
        ! H(H2O)-mim:
                W(6,7)=0.0
        ! athermal
        !       W(1,3)=0.
        !       W(1,4)=0.
        !       W(2,3)=0.   etc.
                do 1400 I=1,NG-1
                do 1400 J=I+1,NG
         1400   W(J,I)=W(I,J)
                call param(T,TT)
            DO 1000 I=1,NK
                R(I)=0.0
                Q(I)=0.0
                GP(I)=0.0
            DO 1000 J=1,NG
                QS(I,J)=RS(I,J)-2.0*(BLIS(I,J)+RS(I,J)-1.0)/ZZ
                R(I)=R(I)+RS(I,J)
         1000 Q(I)=Q(I)+QS(I,J)
            DO 1500 I=1,NK
            DO 1600 J=1,NG
         1600 AL(J)=QS(I,J)/Q(I)
            CALL IKS
            DO 1500 J=1,NG
         1500 GP(I)=GP(I)-ZZ*QS(I,J)*DLOG(XX(J))
        ! for debugging: test call
        !       X1(1)=0.5
        !       X1(2)=0.5
        !       X1(3)=0.0
        !       call ACTCF(X1,GAM1)
```

```
!          continue
           return
           end
! Quasichem  Free Energy
REAL*8 FUNCTION add(n,unrept,unterm)
Implicit REAL*8 (A-H,O-Z)
           add=(n-1.0)*unrept+unterm
RETURN
END
!***********************************************************************
SUBROUTINE PARAM(T,TT)
IMPLICIT REAL*8 (A-H,O-Z)
COMMON /B/ W(10,10),WT(10,10),WTT(10,10),REDT(10,10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
           C1=TT/T
   C2=C1-1.0
   C1=DLOG(C1)-C2
DO 5 I=1,NG
DO 5 J=1,NG
  5  ET(I,J)=DEXP(-W(I,J)-WT(I,J)*C2-WTT(I,J)*C1)
RETURN
END
!***********************************************************************
SUBROUTINE IKS
IMPLICIT  REAL*8 (A-H,O-Z)
REAL*8 RXX(10)
COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
           RE=1.0
   J1=0
DO 1 K=1,NG
   XX(K)=RE
  1  RE=RE-1.0D-04
 19  J1=J1+1
DO 2 K=1,NG
  2  XX(K)=DABS(XX(K))
DO 3 K=1,NG
     BB=0.0
DO 4 I=1,NG
   4 BB=BB+AL(I)*XX(I)*ET(K,I)
  3  RXX(K)=1.0/BB
DO 6 K=1,NG
```

```
  6  XX(K)=(XX(K)+RXX(K))/2.0
DO 7 K=1,NG
IF (DABS(XX(K)-RXX(K)).GT.1.0D-12) GO TO 19
  7  CONTINUE
 20  RETURN
END
!**********************************************************************
Subroutine shell(shape,XA,Rc,gsh)
! input: shape - cyl or sphere
!                    XA -core composition
!                    Rc- core radius
! output gsh - shell free energy
IMPLICIT REAL*8 (A-H,O-Z)
CHARACTER*3 shape
COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
COMMON /B/ W(10,10),WT(10,10),WTT(10,10),REDT(10,10)
COMMON /C/ QT,RT,T,TT,R(10),RS(10,10),Q(10),QS(10,10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
COMMON /E/ P,CP,CV,CO,RO
COMMON /Z/ UM,VEC,HEC,GMC,GGA(10)
COMMON /Y/ H,VheadA,VheadB,VtailA,VtailB,CH2lgth
common /out/ ldamp
if(shape.eq.'sph') k=2.0
if(shape.eq.'cyl') k=1.0
          FiA=XA*fisurf(Rc,k,H,VheadA,VtailA)
          FiB=(1.0-XA)*fisurf(Rc,k,H,VheadB,VtailB)
          !FiA=0.0
          fisolv=1.0-FiA-FiB
! calculate mol fractions from volume fractions
          XNR=FiA/R(1)+FiB/R(2)+fisolv/R(3)
          !write (*,*) fisolv,FiA,FiB
          X(3)=fisolv/(R(3)*XNR)
          X(2)=FiB/(R(2)*XNR)
          X(1)=FiA/(R(1)*XNR)
          gloc=ANCOR (Rc,k,H,CH2lgth)
call bondmixshell(gmxbond)
          gsh=gmxbond     !+gloc
if (ldamp.GT.0) then
if(shape.eq.'sph') write (15,'(4(2X,F8.4),3(2X,E12.5))') XA,FiA,FiB,fisolv,gloc, gmxbond, gsh
if(shape.eq.'cyl') write (12,'(4(2X,F8.4),3(2X,E12.5))') XA,FiA,FiB,fisolv,gloc, gmxbond, gsh
endif
return
end
```

```
!**********************************************************************
SUBROUTINE bondmixshell(gmxbond)
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 QREM(10),XREM(10),GPR(10),QSREM(10,10),RREM(10),XXRE(10)
COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
COMMON /B/ W(10,10),WT(10,10),WTT(10,10),REDT(10,10)
COMMON /C/ QT,RT,T,TT,R(10),RS(10,10),Q(10),QS(10,10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
COMMON /E/ P,CP,CV,CO,RO
COMMON /Z/ UM,VEC,HEC,GMC,GGA(10)
common /out/ ldamp

! store composition and parameters:
          QTR=QT
          DO 111 J=1,NG
 111      XXRE(J)=XX(J)
    DO 1 I=1,NK
    XREM(I)=X(I)
    RREM(I)=R(I)
    QREM(I)=Q(I)
    GPR(I)=GP(I)
    DO 2 J=1,NG
  2   QSREM(I,J)=QS(I,J)
  1   CONTINUE
    NKREM=NK
! infinite dilution X's unsymmetric residual reference act coef.- GP(I)
          X(1)=0.0
          X(2)=0.0
          X(3)=1.0
          DO 1600 I=1,NK
    DO 1600 J=1,NG
 1600 AL(J)=QS(I,J)*X(I)/Q(3)
    CALL IKS
          DO 1500 I=1,NK
          GP(I)=0.0
    DO 1500 J=1,NG
 1500 GP(I)=GP(I)-ZZ*QS(I,J)*DLOG(XX(J))
!         call ACTCF(X,GAM1)
! GGA- act coeff with symmetric reference combinatorial and unsymmetric reference residual parts
! this is OK for solvent
          call ACTSHELL(XREM,GGA)
! for surfactants - correct for bonding and different ref.state in combinatorial, see page11
          AAC=XREM(1)+XREM(2)
```

```
                    gmxbond=0.0
                DO 13 I=1,NK-1
   13           gmxbond=gmxbond+XREM(I)*Dlog( GGA(I) )/AAC
                call ACTCF(XREM,GAM1)
             gmxwater=XREM(3)*Dlog( GAM1(3)*XREM(3) )/AAC
          if (ldamp.GT.0) write (17,'(3(2X,F8.4),3(2X,E12.5))')
XREM(1),XREM(2),XREM(3),gmxbond,gmxwater,(gmxbond+gmxwater)
                gmxbond=gmxbond+gmxwater
          333   format(2X,F10.4,4(4X,E14.6))
             RETURN
             END
          !***********************************************************************
          SUBROUTINE ACTSHELL(DX,GAM)
          ! calulates activity coeff., not LN(Gam) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          IMPLICIT REAL*8 (A-H,O-Z)
          REAL*8 DX(10),GAM(10)
          COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
          COMMON /C/ QT,RT,T,TT,R(10),RS(10,10),Q(10),QS(10,10)
          COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
          COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
             RT=0.0
             QT=0.0
                   !write (*,*) DX(1),DX(2),DX(3),Q(3),R(3)
                   !stop
             DO 1 I=1,NK
             RT=RT+R(I)*DX(I)
           1 QT=QT+Q(I)*DX(I)
             DO 2 I=1,NK
             A=R(3)/RT
             BB=A*QT/Q(3)
           2 GAM(I)=-ZR*Q(I)*DLOG(BB)
             DO 3 L=1,NG
             A=0.0
             DO 4 J=1,NK
           4 A=A+QS(J,L)*DX(J)
           3 AL(L)=A/QT
             CALL IKS
             DO 5 I=1,NK
             DO 6 L=1,NG
           6 GAM(I)=GAM(I)+ZZ*QS(I,L)*DLOG(XX(L))
             GAM(I)=DEXP(GAM(I)+GP(I))
           5 continue
           RETURN
          END
```

```
!**********************************************************************
REAL*8 FUNCTION ANCOR (Rc,k,H,SegL)
    Implicit REAL*8 (A-H,O-Z)
!I-------------------------------------------------------
!I   SegL-segment length, Rc-core radius; k=2 for sphere, k=1 for cylinder
!I-------------------------------------------------------
          k1=k+1.0
          ANCOR=dlog( ((Rc+H)**k1-Rc**k1)/(k1*SegL*Rc**k) )
RETURN
END
!**********************************************************************
SUBROUTINE ACTCF(DX,GAM)
! calulates activity coeff., not LN(Gam)
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 DX(10),GAM(10)
COMMON /S/ X(10),X1(10),X2(10),XX(10),Y(10),GAM1(10),GAM2(10)
COMMON /C/ QT,RT,T,TT,R(10),RS(10,10),Q(10),QS(10,10)
COMMON /D/ ZZ,ZR,NK,NG,JX,IJ
COMMON /H/ BLIS(10,10),AL(10),ALI(10),ET(10,10),GP(10)
    RT=0.0
    QT=0.0
DO 1 I=1,NK
    RT=RT+R(I)*DX(I)
  1  QT=QT+Q(I)*DX(I)
DO 2 I=1,NK
    A=R(I)/RT
    BB=A*QT/Q(I)
  2  GAM(I)=-ZR*Q(I)*(1.0-BB+DLOG(BB))+1.0-A+DLOG(A)
DO 3 L=1,NG
    A=0.0
DO 4 J=1,NK
  4  A=A+QS(J,L)*DX(J)
  3  AL(L)=A/QT
CALL IKS
DO 5 I=1,NK
DO 6 L=1,NG
  6  GAM(I)=GAM(I)+ZZ*QS(I,L)*DLOG(XX(L))
  5  GAM(I)=DEXP(GAM(I)+GP(I))
RETURN
END
!**********************************************************************
REAL*8 FUNCTION fisurf(Rc,k,H,Vhead,Vtail)
Implicit REAL*8 (A-H,O-Z)
!I   Rc-core radius; k=2 for sphere, k=1 for cylinder
```

```
      k1=k+1.0
          VA=H*(1.0+0.5*k*(H/Rc)**2.0+k*(k-1.0)*(H/Rc)**3.0/6.0)
          fisurf=Vhead*Rc/(k1*VA*Vtail)
      RETURN
      END
      !*********************************************************************
```

## Б. Программное обеспечение FORTRAN для расчета в приближении НЛПБ электростатических потенциалов и электростатической свободной энергии заряженной диэлектрической капсулы заданной геометрии, находящейся в водно-солевом растворе

```
      ! Numerical solution of NLPB OUTSIDE and INSIDE of the charged sphercal shell of given radii Rin,Rout [nm] and given dielectric permittivity;
      !surfacte charge density sigma in [C/m^2]    at different salinities;
      !Calculates the electrostatic free energy
      !        S, x0, y0 - reduced charge density, reduced R and reduced el.potential at x0 (the surfaces) ;  dydx0 - derivative at the charged surfaces
          IMPLICIT NONE
          DOUBLE PRECISION S, X0, X_INF, Y0_INIT, NP_FACT, M,   TOL
      C constants declaration
              REAL*8 Pi, e0, k, Na,epsVac, epsW, epsM
          PARAMETER (Pi=3.141592653589793,e0=0.160217733d-18,
         +          k=1.380658d-23,Na = 6.0221367d+23,
              &   epsVac=8.85418d-12, epsW=78.2d0, epsM=2.1d0)
      C other things declaration
          DOUBLE PRECISION F_el, y0, Y0_Y_INF, dydx0
              REAL*8 T, R, sigma, lgCsalt1, lgCsalt2, lgCsalt,
         +  l_B, fact1, fact2, kappa
          real*8 lc,vstar,tauin,tauout,tau,Rin,Rout,Naggr,NagIN,NagOUT,
         & sigmaIN,sigmaOUT,alp_in,alp_out,S_chden,Qfix_in,Qfix_out,
              & Qeff_in,Qeff_out,u_inREM,lckap,M2,S2,NC1,NC2,v_tail,
              & Qmob_out,Qmob_in,u_in,u_out,d_tin,d_tout,d_u,Lapla,x_in,xx,dxx
          LOGICAL DUMP
              COMMON M2,S2,lckap,DUMP
          external FCN_EQN,FCN_JAC,FCN_BCIN,FCN_BCOUT,Naggr,S_chden,     ! FCNPEQ, FCNPBC
              & Lapla,v_tail
      C code section
          T=298.16d0
              DUMP=.FALSE.
      !         R=3.0d-9                                                    ! [m]
```

```
!           sigma=0.025d0
            ! C/m^2
            l_B=e0**2/(4.0d0*Pi*epsW*epsVac*k*T)                    ! [m]  NOTE  we use here temperature
independent epsW    (0.542090973336016e-7*dexp(0.46e-2*T)/T)
            fact2=4.0d0*Pi*l_B/e0
        fact1=dsqrt(8.0d0*Pi*10**3*Na*l_B)


            NC1=8.0d0
             ! C8-surfactant
            NC2=8.0d0
!           vstar =(54.6d0+(NC2-1.0d0)*26.9d0)*1.0d-30              ![A^3]   --->  [m^3]
        lc =(1.5d0+NC2*1.265d0)*1.0d-10                                    ![A]   ---> [m]
            tauin=0.6d0*lc
!           inner thickness of bilayer
            tauout=0.6d0*lc
!           outer thickness of bilayer
            alp_in=0.52d0
! charge fractions in the inner and outer layers
            alp_out=0.56d0
            tau=tauin+tauout                                                ! [m]
            Rin=5.0d0*lc                                                        ! [m]
              GIVE  RIN
            Rout=Rin+tau
            vstar=alp_in*v_tail(NC1)+(1.0d0-alp_in)*v_tail(NC1)
            NagIN = Naggr((Rin+tauin),Rin,vstar)
            vstar=alp_out*v_tail(NC1)+(1.0d0-alp_out)*v_tail(NC1)
            NagOUT= Naggr(Rout,(Rin+tauin),vstar)
            sigmaIN = S_chden(alp_in,NagIN,Rin)                     ! [C/m^2]  fixed inner charge -
surf charge density at the inner membrane
            sigmaOUT = S_chden(alp_out,NagOUT,Rout)                 ! [C/m^2]  fixed outer charge -
surf charge density at the outer membrane
            Qfix_in = 4.0d0*Pi*Rin**2*sigmaIN/e0          !  fixed charge in electrons
            Qfix_out= 4.0d0*Pi*Rout**2*sigmaOUT/e0
            write(*,*) Qfix_in,Qfix_out,(Qfix_in+Qfix_out)



        lgCsalt1=dlog10(1.0d-6)                                             ! [M]
        lgCsalt2=dlog10(6.0d0)
        lgCsalt1=dlog10(1.0d-5)                                             ! [M]
            lgCsalt = lgCsalt1
            X_INF = 20D0
            Y0_INIT = 12                                                    !  do not forget about
anionic (put "-"Y0_INIT)  OR   cationic  (put "+"Y0_INIT) vesicle
        M = 2.0D0
            NP_FACT=0.1d0
            TOL = 1.0D-4
        OPEN(unit=20,file='plotin.dat')
```

```
      Open(unit=11,file='laplace.dat')
      Open(unit=12,file='transme.dat')
     WRITE(*,'(1X,2(A9, F10.5))') "sigmaIN=",sigmaIN,"Rin=",Rin*1.0d9
     WRITE(*,'(1X,2(A9, F10.5))') "sigmaOUT=",sigmaOUT,"Rout="
      &,Rout*1.0d9
     WRITE(20,'(8X, 6(A8,6X),3(A8,3X))') "lgCsalt ","S      "," x0   "
    &," y0     ","F_el  ","Y0_Y_INF","delta_S","Qfix+mob","Qfix"

      do lgCsalt = lgCsalt1, lgCsalt2, 0.1
     DUMP=.FALSE.
      u_inREM = Y0_INIT
!     sigma=2.1d0*sigmaIN
                ! effective charge seen from inner
     sigma=1.1d0*sigmaIN
                ! effective charge seen from inner
       kappa=fact1*dsqrt(10**lgCsalt)
       lckap=lc*kappa
     s=fact2*sigma/kappa
  112  x0=kappa*Rin
     Qeff_in=s*Rin**2*kappa/l_B                    ! effective charge seen from inner
      CALL PB_NUM_IN(S, X0, X_INF, Y0_INIT, NP_FACT, M, TOL, F_el, y0,  ! inner domain
calculation
    +  Y0_Y_INF, dydx0,Qmob_in)
      Qmob_in=4.0d0*Pi*Rin**2*Qmob_in*Na*(10**lgCsalt)*1.0d3/kappa
       d_tin=s-fact2*sigmaIN/kappa
       d_u = d_tin*(epsW/epsM)*(Rin/Rout)*(Rin-Rout)*kappa                    !
transmembrane potential
       u_out = y0-d_u
       u_in = y0
       x_in =x0
       d_tout = - d_tin*(Rin/Rout)**2
         s=fact2*sigmaOUT/kappa                    !fixed outer charge
         s  = d_tout+s                    ! effective charge seen from outer
       Qeff_out=s*Rout**2*kappa/l_B
          x0=kappa*Rout
      CALL PB_NUM_OUT(S, X0, X_INF, Y0_INIT, NP_FACT,M, TOL, F_el, y0,
    +  Y0_Y_INF, dydx0,Qmob_out)
       Qmob_out=4.0d0*Pi*Rout**2*Qmob_out*Na*(10**lgCsalt)*1.0d3/kappa
       write(*,*) 'potentials ',(u_out-y0),d_u,(u_in-y0),u_in
       write(*,*) 'zero?', dydx0+s

       if (dabs(u_out-y0).GT.1.0d-5.OR.dabs(u_in-u_inREM).GT.1.0d-5)then
       u_inREM=u_in
       d_u=u_in-y0
       s = d_u*(epsM/epsW)*(Rout/Rin)/((Rin-Rout)*kappa)+
```

```
      & fact2*sigmaIN/kappa
       goto 112
      else                                                    ! print out the result
      DUMP=.TRUE.
      d_u=u_in-y0
      s = d_u*(epsM/epsW)*(Rout/Rin)/((Rin-Rout)*kappa)+
& fact2*sigmaIN/kappa
 Qeff_in=s*Rin**2*kappa/l_B
      x0=kappa*Rin
      CALL PB_NUM_IN(S, X0, X_INF, Y0_INIT, NP_FACT, M, TOL, F_el, y0,
& Y0_Y_INF, dydx0,Qmob_in)
 Qmob_in=4.0d0*Pi*Rin**2*Qmob_in*Na*(10**lgCsalt)*1.0d3/kappa
       WRITE(20,'(9(F14.5))') lgCsalt, S, x0, y0, F_el,
& Y0_Y_INF, (dydx0-s),(Qfix_in+Qmob_in),Qfix_in

      s=d_tout+fact2*sigmaOUT/kappa
      Qeff_out=s*Rout**2*kappa/l_B
         x0=kappa*Rout
  CALL PB_NUM_OUT(S, X0, X_INF, Y0_INIT, NP_FACT,M, TOL, F_el, y0,
&  Y0_Y_INF, dydx0,Qmob_out)
      write(*,*) 'zero?', dydx0+s
       Qmob_out=4.0d0*Pi*Rout**2*Qmob_out*Na*(10**lgCsalt)*1.0d3/kappa
      WRITE(20,'(9(F14.5))') lgCsalt, S, x0, y0, F_el,
      &Y0_Y_INF, (dydx0+s),(Qfix_out+Qmob_out),Qfix_out
       write (12,'(F14.2,F14.8,F14.4))') Rin/lc, (10**lgCsalt), d_u
      endif

      write(*,*) 'netrality,% ',(Qeff_out+Qeff_in-Qfix_in-Qfix_out),
&(Qeff_out-Qfix_out-Qfix_in-Qmob_in),
      &(Qeff_in-Qfix_out-Qfix_in-Qmob_out),
      &(1.d0+(Qmob_out+Qmob_in)/(Qfix_out+Qfix_in))*100
      write(*,*) 'eff. charge IN OUT',Qeff_in,Qeff_out
 dxx=(x0-x_in)/2.0d+1
      xx=x_in
      do while (xx.LE.x0)
      write(11,'(1X, 2(F14.7,3X))') xx/lckap, Lapla(xx,u_in,y0,x_in,x0)
      xx = xx + dxx
      enddo
 enddo                                          !end of salt-cycle
      close(20)
      close(11)
      close (12)
      stop
      END
```

```
C*************************************************************************************
      SUBROUTINE PB_NUM_IN(S, X0, X_INF, Y0_INIT, NP_FACT, M, TOL, F_el,
     + y0, Y0_Y_INF,dydx0,Qmob)
C
C     This subroutine uses IMSL library to solve one-dimentional Poisson-Boltzamnn equation
C                    d2y/dx2+m/x*dy/dx=sinh(y)
C  between x=0 and x=x0 with boundary conditions dy/dx=s at x=x0 and dy/dx=0 at x=0.
C
C  On entry:
C  S - reduced charge density
C  X0 - reduced surface coordimate
C  X_INF - effective infinity in reduced units
C  Y0_INIT - initial approximation for reduced surface potential
C  M - shape-dependent parameter
C   NP_FACT - ratio of initial number of mesh points to the maximal number of mesh points, 0 < NP_FACT
< 1
C  TOL - absolute tolerance for solution
C  On exit:
C  F_el - electrostatic free energy per elementary charge
C     y0 - surface potential
C  Y0_Y_INF - ratio of potential at infinity to that at the surface
C  dydx0 - actual value of dy/dx at x=x0, must be s+dydx0=0
      IMPLICIT NONE
      INTEGER N,MNP,LIW, LW,NINIT,NFINAL,LDYINI,LDYFIN,MXGRID
      PARAMETER     (N=2,MNP=100000,LW=MNP*(3*N*N+6*N+2)+4*N*N+4*N, !     2N(3N     *
MXGRID + 4N + 1) + 2 * MXGRID(7N + 2) + 2N * MXGRID + N + MXGRID
     &          LIW=MNP*(2*N+1)+N*N+4*N+2,LDYINI=N,LDYFIN=N)
      Real*8 S, X0, X_INF, Y0_INIT, NP_FACT, M, TOL
      Real*8 F_el, y0, Y0_Y_INF, dydx0,Qmob,Q
*     .. Scalars in Common ..
      Real*8 M2,S2
      Logical linear,print1,DUMP
        Real*8 F, F0, SUM0,SUM1,lckap
      Real*8 A, B, LOG_A, LOG_B, STEP,TLEFT,TRIGHT,PISTEP
      INTEGER I,J,NP,NLEFT,NCUPBC
        Real*8 X(MNP),YINIT(N,MNP),YFINAL(N,MNP),TFINAL(MNP),ERREST(N)
!     INTEGER     IW(LIW)
*     .. Intrinsic Functions ..
      INTRINSIC   DBLE
      COMMON      M2,S2,lckap,DUMP
      external FCN_EQN,FCN_JAC,FCN_BCIN     ! FCNPEQ, FCNPBC

      OPEN(UNIT=10,FILE='solution.dat')
        M2=M
```

```
            S2=S
        NP = INT(NP_FACT*MNP)

        A = X0
            B = X0+X_INF

    C    MESH SECTION
    !    y0_init
            X(1)=1.0d-08
            step = A/dble(np-1)
             do i=2,np-1
            X(i)= x(1)+(i-1)*step
             enddo
        X(NP) = A
    !       yinit(1,NP) = y0_init
        do i=1,np
            yinit(1,i)=y0_init     !0.001d0*S*(dsinh(x(i))/x(i))*A**2/(A*dcosh(A)-dsinh(A))
    !       WRITE(10,*) i, X(i), yinit(1,i)
    !       WRITE(10,*) i, X(i),
    !   &   S*(dsinh(x(i))/x(i))*A**2/(A*dcosh(A)-dsinh(A))
            enddo
    C
    C    SOLUTION SECTION
    !       x(1) =1.0d0-8
            NLEFT  = 1                          ! number of boundary conditions  at the left side
        NCUPBC = 0                      ! number of bnd conditions involving both sides
        TOL    = .001           ! relative error control
        TLEFT  = x(1)
        TRIGHT = x(np)
        PISTEP = 0.0
        PRINT1  = .false.
        LINEAR = .FALSE.
            NINIT = NP
            MXGRID = MNP

             CALL DBVPFD (FCN_EQN,fcn_jac,fcn_bcin,FCN_EQN,FCN_BCIN, N,
       &     NLEFT, NCUPBC, TLEFT, TRIGHT, PISTEP, TOL,
       &     NINIT, X, YINIT, LDYINI, LINEAR, PRINT1,
       &     MXGRID, NFINAL, TFINAL, YFINAL, LDYFIN, ERREST)
    ! FCNEQN — User-supplied SUBROUTINE to evaluate derivatives. The usage is CALL FCNEQN (N, T, Y, P, DYDT), where
    !N — Number of differential equations.   (Input)
    !T — Independent variable, t.   (Input)
    !Y — Array of size N containing the dependent variable values, y(t).   (Input)
```

!P — Continuation parameter, p. (Input)

!DYDT — Array of size N containing the derivatives yў(t). (Output)

!The name FCNEQN must be declared EXTERNAL in the calling program.

!FCNJAC — User-supplied SUBROUTINE to evaluate the Jacobian. The usage is CALL FCNJAC (N, T, Y, P, DYPDY), where

!N — Number of differential equations. (Input)

!T — Independent variable, t. (Input)

!Y — Array of size N containing the dependent variable values. (Input)

!P — Continuation parameter, p. (Input)

!DYPDY — N by N array containing the partial derivatives ai,j = dfi/dyj evaluated at (t, y). The values ai,j are returned in DYPDY(i, j). (Output)

!The name FCNJAC must be declared EXTERNAL in the calling program.

!FCNBC — User-supplied SUBROUTINE to evaluate the boundary conditions. The usage is CALL FCNBC (N, YLEFT, YRIGHT, P, H), where

!N — Number of differential equations. (Input)

!YLEFT — Array of size N containing the values of the dependent variable at the left endpoint. (Input)

!YRIGHT — Array of size N containing the values of the dependent variable at the right endpoint. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!H — Array of size N containing the boundary condition residuals. (Output)

!The boundary conditions are defined by hi = 0; for i = 1, . . . , N. The left endpoint conditions must be defined first, then, the conditions involving both endpoints, and finally the right endpoint conditions.

!The name FCNBC must be declared EXTERNAL in the calling program.

!FCNPEQ — User-supplied SUBROUTINE to evaluate the partial derivative of yў with respect to the parameter p. The usage is CALL FCNPEQ (N, T, Y, P, DYPDP), where

!N — Number of differential equations. (Input)

!T — Dependent variable, t. (Input)

!Y — Array of size N containing the dependent variable values. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!DYPDP — Array of size N containing the partial derivatives ai,j = ¶fi /¶yj evaluated at (t, y). The values ai,j are returned in DYPDY(i, j). (Output)

!The name FCNPEQ must be declared EXTERNAL in the calling program.

!FCNPBC — User-supplied SUBROUTINE to evaluate the derivative of the boundary conditions with respect to the parameter p. The usage is CALL FCNPBC (N, YLEFT, YRIGHT, P, H), where

!N — Number of differential equations. (Input)

!YLEFT — Array of size N containing the values of the dependent variable at the left endpoint. (Input)

!YRIGHT — Array of size N containing the values of the dependent variable at the right endpoint. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!H — Array of size N containing the derivative of fi with respect to p. (Output)

!The name FCNPBC must be declared EXTERNAL in the calling program.

!N — Number of differential equations. (Input)

!NLEFT — Number of initial conditions. (Input)

!The value NLEFT must be greater than or equal to zero and less than N.

!NCUPBC — Number of coupled boundary conditions. (Input)

!The value NLEFT + NCUPBC must be greater than zero and less than or equal to N.

!TLEFT — The left endpoint. (Input)

!TRIGHT — The right endpoint. (Input)

!PISTEP — Initial increment size for p. (Input)

!If this value is zero, continuation will not be used in this problem. The routines FCNPEQ and FCNPBC will not be called.

!TOL — Relative error control parameter. (Input)

!The computations stop when ABS(ERROR(J, I))/MAX(ABS(Y(J, I)), 1.0).LT.TOL for all J = 1, . . . , N and I = 1, . . . , NGRID. Here, ERROR(J, I) is the estimated error in Y(J, I).

!NINIT — Number of initial grid points, including the endpoints. (Input)

!It must be at least 4.

!TINIT — Array of size NINIT containing the initial grid points. (Input)

!YINIT — Array of size N by NINIT containing an initial guess for the values of Y at the points in TINIT. (Input)

!LDYINI — Leading dimension of YINIT exactly as specified in the dimension statement of the calling program. (Input)

!LINEAR — Logical .TRUE. if the differential equations and the boundary conditions are linear. (Input)

!PRINT — Logical .TRUE. if intermediate output is to be printed. (Input)

!MXGRID — Maximum number of grid points allowed. (Input)

!NFINAL — Number of final grid points, including the endpoints. (Output)

!TFINAL — Array of size MXGRID containing the final grid points. (Output)

!Only the first NFINAL points are significant.

!YFINAL — Array of size N by MXGRID containing the values of Y at the points in TFINAL. (Output)

!LDYFIN — Leading dimension of YFINAL exactly as specified in the dimension statement of the calling program. (Input)

!ERREST — Array of size N. (Output)

!ERREST(J) is the estimated error in Y(J).

!2. Informational errors

| !Type | Code | |
|---|---|---|
| ! 4 | 1 | More than MXGRID grid points are needed to solve the problem. |
| ! 4 | 2 | Newton's method diverged. |
| ! 3 | 3 | Newton's method reached roundoff error level. |

!3. If the value of PISTEP is greater than zero, then the routine BVPFD assumes that the user has embedded the problem into a one-parameter family of problems:

!yў = yў(t, y, p)

!h(ytleft, ytright, p) = 0

!such that for p = 0 the problem is simple. For p = 1, the original problem is recovered. The routine BVPFD automatically attempts to increment from p = 0 to p = 1. The value PISTEP is the beginning increment used in this continuation. The increment will usually be changed by routine BVPFD, but an arbitrary minimum of 0.01 is imposed.

```
      Y0=YFINAL(1,NFINAL)
      Y0_Y_INF=YFINAL(1,1)/YFINAL(1,NFINAL)
    dydx0=YFINAL(2,NFINAL)
    WRITE(*,'(1X,A8,2(2X,F14.7))')'dy/dx|0=',
     &                YFINAL(1,NFINAL),YFINAL(2,NFINAL)
    if (DUMP) WRITE (10,'(1X, 2(F14.7, 5X), F14.7)')                    ! writing to file when 1
     &     (X(I)/lckap,(YFINAL(J,I),J=1,N),I=1,NFINAL)
C    INTEGRATION SECTION
C
      SUM0=0.0D0
      SUM1=0.0d0
      I=1
      Q = DSINH(YFINAL(1,I))*(X(I)/A)**M

      F0=(0.5D0*YFINAL(2,I)**2+YFINAL(1,I)*DSINH(YFINAL(1,I))
     &-DCOSH(YFINAL(1,I))+1.0d0)*(X(I)/A)**M
      DO I=2,NFINAL
       F=(0.5D0*YFINAL(2,I)**2+YFINAL(1,I)*DSINH(YFINAL(1,I))
     &-DCOSH(YFINAL(1,I))+1.0d0)*(X(I)/A)**M
        SUM0=SUM0+(X(I)-X(I-1))*(F+F0)
        F0=F
        Qmob=          DSINH(YFINAL(1,I))*(X(I)/A)**M
        SUM1= SUM1+(X(I)-X(I-1))*(Qmob+Q)
        Q = Qmob
      ENDDO
      Qmob=-SUM1
      F_el=SUM0/(2.0D0*S)
    CLOSE(UNIT=10)
    END
C*******************************************************************************
    subroutine FCN_BCIN(N, YLEFT, YRIGHT, P, H)
      implicit none
      integer N
      real*8 YLEFT(N),YRIGHT(N),H(N),P,M,S
      Common  M,S
!H — Array of size N containing the boundary condition residuals.   (Output)
!The boundary conditions are defined by hi = 0; for i = 1,  . . . , N. The left endpoint conditions must be defined
first,
!then, the conditions involving both endpoints, and finally the right endpoint conditions.
!The name FCNBC must be declared EXTERNAL in the calling program.
       H(1)=YLEFT(2)          !left boundary      - derivative
       H(2)=YRIGHT(2)-S              !Y2|inf=0  derivative
      return
      end
```

```
C*****************************************************************************
      real*8 function Lapla(x,uin,uout,xin,xout)
            implicit none
            real*8 x,uin,uout,xin,xout
            Lapla = xout*(uin-uout)*(1.0d0-xin/x)/(xin-xout)+uin
            return
            end function
C*****************************************************************************
      SUBROUTINE PB_NUM_OUT(S,X0, X_INF, Y0_INIT,NP_FACT, M, TOL, F_el, + y0, Y0_Y_INF, dydx0,
Qmob)
C     This subroutine uses IMSL library to solve one-dimentional Poisson-Boltzamnn equation
C                        d2y/dx2+m/x*dy/dx=sinh(y)
C   between x=x0 and x=infinity with boundary conditions dy/dx=-s at x=x0 and dy/dx=0 at x=infinity.
C
C   On entry:
C   S - reduced charge density
C   X0 - reduced surface coordimate
C   X_INF - effective infinity in reduced units
C   Y0_INIT - initial approximation for reduced surface potential
C   M - shape-dependent parameter
C    NP_FACT - ratio of initial number of mesh points to the maximal number of mesh points, 0 < NP_FACT
< 1
C   TOL - absolute tolerance for solution
C   On exit:
C   F_el - electrostatic free energy per elementary charge
C       y0 - surface potential
C   Y0_Y_INF - ratio of potential at infinity to that at the surface
C   dydx0 - actual value of dy/dx at x=x0, must be s+dydx0=0
      IMPLICIT NONE
      INTEGER N,MNP,LIW, LW,NINIT,NFINAL,LDYINI,LDYFIN,MXGRID
      PARAMETER     (N=2,MNP=10000,LW=MNP*(3*N*N+6*N+2)+4*N*N+4*N, !        2N(3N      *
MXGRID + 4N + 1) + 2 * MXGRID(7N + 2) + 2N * MXGRID + N + MXGRID
     &            LIW=MNP*(2*N+1)+N*N+4*N+2,LDYINI=N,LDYFIN=N)
      Real*8 S, X0, X_INF, Y0_INIT, NP_FACT, M, TOL
      Real*8 F_el, y0, Y0_Y_INF, dydx0
*     .. Scalars in Common ..
      Real*8 M2,S2,lckap
      Logical linear,print1,DUMP
            Real*8 F, F0, SUM0, SUM1, Qmob,Q
      Real*8 A, B, LOG_A, LOG_B, STEP,TLEFT,TRIGHT,PISTEP
      INTEGER  I, J, NP,NLEFT,NCUPBC

            Real*8 X(MNP),YINIT(N,MNP),YFINAL(N,MNP),TFINAL(MNP),ERREST(N)
!     INTEGER       IW(LIW)
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        DBLE
      COMMON           M2,S2,lckap,DUMP
      external FCN_EQN,FCN_JAC,FCN_BCOUT     ! FCNPEQ, FCNPBC

      OPEN(UNIT=10,FILE='solution.dat',access='append')
            M2=M
            S2=S
      NP = INT(NP_FACT*MNP)

      A = X0
            B = X0+X_INF
C     MESH SECTION
C
C       X(1) = A
C       DO I = 2, NP - 1
C         X(I) = (DBLE(NP-I)*A+DBLE(I-1)*B)/DBLE(NP-1)
C       ENDDO
C       X(NP) = B

            LOG_A = DLOG(A)
            LOG_B = DLOG(B)
            STEP = (LOG_B-LOG_A)/DBLE(NP-1)
      X(1) = A
            yinit(1,1) =y0_init
      DO I = 2, NP - 1
        X(I)   = DEXP(LOG_A + (I-1)*STEP)
             yinit(1,i) =y0_init*dexp(A-x(i))
C           WRITE(10,*) I, X(I), yinit(1,i)
      ENDDO
      X(NP) = B
            yinit(1,NP) = 0.0d0

C     SOLUTION SECTION
            NLEFT  = 1                          ! number of boundary conditions  at the left side
      NCUPBC = 0                        ! number of bnd conditions involving both sides
      TOL    = .001             ! relative error control
      TLEFT  = A
      TRIGHT = B
      PISTEP = 0.0
      PRINT1  = .FALSE.
      LINEAR = .FALSE.
            NINIT = NP
            MXGRID = MNP
```

```
      CALL DBVPFD (FCN_EQN,fcn_jac,fcn_bcout,FCN_EQN,FCN_BCOUT, N,
     &    NLEFT, NCUPBC, TLEFT, TRIGHT, PISTEP, TOL,
     &    NINIT, X, YINIT, LDYINI, LINEAR, PRINT1,
     &    MXGRID, NFINAL, TFINAL, YFINAL, LDYFIN, ERREST)
```

! FCNEQN — User-supplied SUBROUTINE to evaluate derivatives. The usage is CALL FCNEQN (N, T, Y, P, DYDT), where

!N — Number of differential equations. (Input)

!T — Independent variable, t. (Input)

!Y — Array of size N containing the dependent variable values, y(t). (Input)

!P — Continuation parameter, p. (Input)

!DYDT — Array of size N containing the derivatives yў(t). (Output)

!The name FCNEQN must be declared EXTERNAL in the calling program.

!FCNJAC — User-supplied SUBROUTINE to evaluate the Jacobian. The usage is CALL FCNJAC (N, T, Y, P, DYPDY), where

!N — Number of differential equations. (Input)

!T — Independent variable, t. (Input)

!Y — Array of size N containing the dependent variable values. (Input)

!P — Continuation parameter, p. (Input)

!DYPDY — N by N array containing the partial derivatives ai,j = dfi/dyj evaluated at (t, y). The values ai,j are returned in DYPDY(i, j). (Output)

!The name FCNJAC must be declared EXTERNAL in the calling program.

!FCNBC — User-supplied SUBROUTINE to evaluate the boundary conditions. The usage is CALL FCNBC (N, YLEFT, YRIGHT, P, H), where

!N — Number of differential equations. (Input)

!YLEFT — Array of size N containing the values of the dependent variable at the left endpoint. (Input)

!YRIGHT — Array of size N containing the values of the dependent variable at the right endpoint. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!H — Array of size N containing the boundary condition residuals. (Output)

!The boundary conditions are defined by hi = 0; for i = 1, . . . , N. The left endpoint conditions must be defined first, then, the conditions involving both endpoints, and finally the right endpoint conditions.

!The name FCNBC must be declared EXTERNAL in the calling program.

!FCNPEQ — User-supplied SUBROUTINE to evaluate the partial derivative of yў with respect to the parameter p. The usage is CALL FCNPEQ (N, T, Y, P, DYPDP), where

!N — Number of differential equations. (Input)

!T — Dependent variable, t. (Input)

!Y — Array of size N containing the dependent variable values. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!DYPDP — Array of size N containing the partial derivatives ai,j = ¶fi /¶yj evaluated at (t, y). The values ai,j are returned in DYPDY(i, j). (Output)

!The name FCNPEQ must be declared EXTERNAL in the calling program.

!FCNPBC — User-supplied SUBROUTINE to evaluate the derivative of the boundary conditions with respect to the parameter p. The usage is CALL FCNPBC (N, YLEFT, YRIGHT, P, H), where

!N — Number of differential equations. (Input)

!YLEFT — Array of size N containing the values of the dependent variable at the left endpoint. (Input)

!YRIGHT — Array of size N containing the values of the dependent variable at the right endpoint. (Input)

!P — Continuation parameter, p. (Input)

!See Comment 3.

!H — Array of size N containing the derivative of fi with respect to p. (Output)

!The name FCNPBC must be declared EXTERNAL in the calling program.

!N — Number of differential equations. (Input)

!NLEFT — Number of initial conditions. (Input)

!The value NLEFT must be greater than or equal to zero and less than N.

!NCUPBC — Number of coupled boundary conditions. (Input)

!The value NLEFT + NCUPBC must be greater than zero and less than or equal to N.

!TLEFT — The left endpoint. (Input)

!TRIGHT — The right endpoint. (Input)

!PISTEP — Initial increment size for p. (Input)

!If this value is zero, continuation will not be used in this problem. The routines FCNPEQ and FCNPBC will not be called.

!TOL — Relative error control parameter. (Input)

!The computations stop when ABS(ERROR(J, I))/MAX(ABS(Y(J, I)), 1.0).LT.TOL for all J = 1, . . . , N and I = 1, . . . , NGRID. Here, ERROR(J, I) is the estimated error in Y(J, I).

!NINIT — Number of initial grid points, including the endpoints. (Input)

!It must be at least 4.

!TINIT — Array of size NINIT containing the initial grid points. (Input)

!YINIT — Array of size N by NINIT containing an initial guess for the values of Y at the points in TINIT. (Input)

!LDYINI — Leading dimension of YINIT exactly as specified in the dimension statement of the calling program. (Input)

!LINEAR — Logical .TRUE. if the differential equations and the boundary conditions are linear. (Input)

!PRINT — Logical .TRUE. if intermediate output is to be printed. (Input)

!MXGRID — Maximum number of grid points allowed. (Input)

!NFINAL — Number of final grid points, including the endpoints. (Output)

!TFINAL — Array of size MXGRID containing the final grid points. (Output)

!Only the first NFINAL points are significant.

!YFINAL — Array of size N by MXGRID containing the values of Y at the points in TFINAL. (Output)

!LDYFIN — Leading dimension of YFINAL exactly as specified in the dimension statement of the calling program. (Input)

!ERREST — Array of size N. (Output)

!ERREST(J) is the estimated error in Y(J).

!2. Informational errors

!Type Code

! 4 1 More than MXGRID grid points are needed to solve the problem.

```
! 4        2       Newton's method diverged.
! 3        3       Newton's method reached roundoff error level.
!3.      If the value of PISTEP is greater than zero, then the routine BVPFD assumes that the user has embedded the problem into a one-parameter family of problems:
!yў = yў(t, y, p)
!h(ytleft, ytright, p) = 0
!such that for p = 0 the problem is simple. For p = 1, the original problem is recovered. The routine BVPFD automatically attempts to increment from p = 0 to p = 1. The value PISTEP is the beginning increment used in this continuation. The increment will usually be changed by routine BVPFD, but an arbitrary minimum of 0.01 is imposed.

      Y0=YFINAL(1,1)
          Y0_Y_INF=YFINAL(1,NFINAL)/YFINAL(1,1)
      dydx0=YFINAL(2,1)
      WRITE(*,'(1X, A8,2(2X,F14.7))')'dy/dx|0=',YFINAL(1,1),YFINAL(2,1)

      if(DUMP) WRITE (10,'(1X, 2(F14.7, 5X), F14.7)')                ! writing to file when 1
     +     (X(I)/lckap,(YFINAL(J,I),J=1,N),I=1,NFINAL)
C     INTEGRATION SECTION
          SUM0=0.0D0
          SUM1=0.0d0
          I=1
          Q = DSINH(YFINAL(1,I))*(X(I)/X(1))**M

          F0=(0.5D0*YFINAL(2,I)**2+YFINAL(1,I)*DSINH(YFINAL(1,I))
     &-DCOSH(YFINAL(1,I))+1)*(X(I)/X(1))**M
          DO I=2,NFINAL
       F=(0.5D0*YFINAL(2,I)**2+YFINAL(1,I)*DSINH(YFINAL(1,I))
     &-DCOSH(YFINAL(1,I))+1)*(X(I)/X(1))**M
           SUM0=SUM0+(X(I)-X(I-1))*(F+F0)
           F0=F
           Qmob=DSINH(YFINAL(1,I))*(X(I)/X(1))**M
           SUM1=SUM1+(X(I)-X(I-1))*(Qmob+Q)
           Q = Qmob
          ENDDO
          Qmob = -SUM1
          F_el=SUM0/(2.0D0*S)
      CLOSE(UNIT=10)
          return
      END
C*******************************************************************************
      subroutine FCN_EQN(N, X, Y, P, DYDX)
          implicit none
          integer N
          real*8 Y(N),DYDX(N),X,P,M,S
```

```
      Common  M,S
      DYDX(1)=Y(2)
      DYDX(2)= DSINH(Y(1))-M*Y(2)/X
      return
      end
C*******************************************************************************
    subroutine FCN_JAC(N, X, Y, P, DYPDY)
      implicit none
      integer N
      real*8 Y(N),DYPDY(N,N),X,P,M,S
      Common  M,S
      DYPDY(1,1)=0.0d0
      DYPDY(1,2)=1.0d0
      DYPDY(2,1)=DCOSH(Y(1))
      DYPDY(2,2)=-M/X
      return
      end
C*******************************************************************************
    subroutine FCN_BCOUT(N, YLEFT, YRIGHT, P, H)
      implicit none
      integer N
      real*8 YLEFT(N),YRIGHT(N),H(N),P,M,S
      Common  M,S
!H — Array of size N containing the boundary condition residuals.   (Output)
!The boundary conditions are defined by hi = 0; for i = 1,  . . . , N. The left endpoint conditions must be defined
first,
!then, the conditions involving both endpoints, and finally the right endpoint conditions.
!The name FCNBC must be declared EXTERNAL in the calling program.
       H(1)=YLEFT(2)+S        !left boundary      - derivative
       H(2)=YRIGHT(2)                  !Y2|inf=0  derivative
      return
      end
C*******************************************************************************
    real*8 function Naggr(Rout,Rin,vstar)
      implicit none
      real*8 vstar, Rin, Rout,Pi
      PARAMETER (Pi=3.141592653589793d0)
      Naggr= 4.0d0*Pi*(Rout**3-Rin**3)/(3.0d0*vstar)
      return
      end function
C*******************************************************************************
    real*8 function S_chden(alp,Nag,R)
      implicit none
      real*8 alp, R, Nag,Pi,e0
```

```
      PARAMETER (Pi=3.141592653589793,e0=0.160217733d-18)
      S_chden= (2.0d0*alp-1.0d0)*Nag*e0/(4.0d0*Pi*R**2)
      return
      end function
C*******************************************************************************
   real*8 function v_tail(NC)
      implicit none
      real*8 NC
      v_tail=(54.6d0+(NC-1.0d0)*26.9d0)*1.0d-30
      return
      end function
C*******************************************************************************
```